Open Source Used In Automated Fault Management 2.0.0

Cisco Systems, Inc.
www.cisco.com

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco website at www.cisco.com/go/offices.

Text Part Number: 78EE117C99-182897582

This document contains licenses and notices for open source software used in this product. With respect to the free/open source software listed in this document, if you have any questions or wish to receive a copy of any source code to which you may be entitled under the applicable free/open source license(s) (such as the GNU Lesser/General Public License), please contact us at external-opensource-requests@cisco.com.

In your requests please include the following reference number 78EE117C99-182897582

Contents

1.1 activemq-client 5.14.5

1.2 antisamy 1.4.3

1.2.1 Available under license :

```
/*
* Copyright (c) 2007-2010, Arshan Dabirsiaghi, Jason Li
*
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
* - Redistributions of source code must retain the above copyright notice,
*       this list of conditions and the following disclaimer.
* - Redistributions in binary form must reproduce the above copyright notice,
*   this list of conditions and the following disclaimer in the documentation
*   and/or other materials provided with the distribution.
* - Neither the name of OWASP nor the names of its contributors may be used to
*   endorse or promote products derived from this software without specific
*   prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

1.3 ANTLR 2.7.7

1.3.1 Available under license :

ANTLR 3 License

[The BSD License]
Copyright (c) 2003-2008, Terence Parr
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
* Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.4 apache-ant_within-cglib 1.6.5

1.4.1 Available under license :

/*
* Apache License
* Version 2.0, January 2004
* http://www.apache.org/licenses/
*
* TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
*
* 1. Definitions.
*
* "License" shall mean the terms and conditions for use, reproduction,
* and distribution as defined by Sections 1 through 9 of this document.
*
* "Licensor" shall mean the copyright owner or entity authorized by
* the copyright owner that is granting the License.
*

```
*      "Legal Entity" shall mean the union of the acting entity and all
*      other entities that control, are controlled by, or are under common
*      control with that entity. For the purposes of this definition,
*      "control" means (i) the power, direct or indirect, to cause the
*      direction or management of such entity, whether by contract or
*      otherwise, or (ii) ownership of fifty percent (50%) or more of the
*      outstanding shares, or (iii) beneficial ownership of such entity.
*
*      "You" (or "Your") shall mean an individual or Legal Entity
*      exercising permissions granted by this License.
*
*      "Source" form shall mean the preferred form for making modifications,
*      including but not limited to software source code, documentation
*      source, and configuration files.
*
*      "Object" form shall mean any form resulting from mechanical
*      transformation or translation of a Source form, including but
*      not limited to compiled object code, generated documentation,
*      and conversions to other media types.
*
*      "Work" shall mean the work of authorship, whether in Source or
*      Object form, made available under the License, as indicated by a
*      copyright notice that is included in or attached to the work
*      (an example is provided in the Appendix below).
*
*      "Derivative Works" shall mean any work, whether in Source or Object
*      form, that is based on (or derived from) the Work and for which the
*      editorial revisions, annotations, elaborations, or other modifications
*      represent, as a whole, an original work of authorship. For the purposes
*      of this License, Derivative Works shall not include works that remain
*      separable from, or merely link (or bind by name) to the interfaces of,
*      the Work and Derivative Works thereof.
*
*      "Contribution" shall mean any work of authorship, including
*      the original version of the Work and any modifications or additions
*      to that Work or Derivative Works thereof, that is intentionally
*      submitted to Licensor for inclusion in the Work by the copyright owner
*      or by an individual or Legal Entity authorized to submit on behalf of
*      the copyright owner. For the purposes of this definition, "submitted"
*      means any form of electronic, verbal, or written communication sent
*      to the Licensor or its representatives, including but not limited to
*      communication on electronic mailing lists, source code control systems,
*      and issue tracking systems that are managed by, or on behalf of, the
*      Licensor for the purpose of discussing and improving the Work, but
*      excluding communication that is conspicuously marked or otherwise
*      designated in writing by the copyright owner as "Not a Contribution."
*
*      "Contributor" shall mean Licensor and any individual or Legal Entity
```

```
*      on behalf of whom a Contribution has been received by Licensor and
*      subsequently incorporated within the Work.
*
*   2. Grant of Copyright License. Subject to the terms and conditions of
*      this License, each Contributor hereby grants to You a perpetual,
*      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
*      copyright license to reproduce, prepare Derivative Works of,
*      publicly display, publicly perform, sublicense, and distribute the
*      Work and such Derivative Works in Source or Object form.
*
*   3. Grant of Patent License. Subject to the terms and conditions of
*      this License, each Contributor hereby grants to You a perpetual,
*      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
*      (except as stated in this section) patent license to make, have made,
*      use, offer to sell, sell, import, and otherwise transfer the Work,
*      where such license applies only to those patent claims licensable
*      by such Contributor that are necessarily infringed by their
*      Contribution(s) alone or by combination of their Contribution(s)
*      with the Work to which such Contribution(s) was submitted. If You
*      institute patent litigation against any entity (including a
*      cross-claim or counterclaim in a lawsuit) alleging that the Work
*      or a Contribution incorporated within the Work constitutes direct
*      or contributory patent infringement, then any patent licenses
*      granted to You under this License for that Work shall terminate
*      as of the date such litigation is filed.
*
*   4. Redistribution. You may reproduce and distribute copies of the
*      Work or Derivative Works thereof in any medium, with or without
*      modifications, and in Source or Object form, provided that You
*      meet the following conditions:
*
*      (a) You must give any other recipients of the Work or
*          Derivative Works a copy of this License; and
*
*      (b) You must cause any modified files to carry prominent notices
*          stating that You changed the files; and
*
*      (c) You must retain, in the Source form of any Derivative Works
*          that You distribute, all copyright, patent, trademark, and
*          attribution notices from the Source form of the Work,
*          excluding those notices that do not pertain to any part of
*          the Derivative Works; and
*
*      (d) If the Work includes a "NOTICE" text file as part of its
*          distribution, then any Derivative Works that You distribute must
*          include a readable copy of the attribution notices contained
*          within such NOTICE file, excluding those notices that do not
*          pertain to any part of the Derivative Works, in at least one
```

```
*       of the following places: within a NOTICE text file distributed
*       as part of the Derivative Works; within the Source form or
*       documentation, if provided along with the Derivative Works; or,
*       within a display generated by the Derivative Works, if and
*       wherever such third-party notices normally appear. The contents
*       of the NOTICE file are for informational purposes only and
*       do not modify the License. You may add Your own attribution
*       notices within Derivative Works that You distribute, alongside
*       or as an addendum to the NOTICE text from the Work, provided
*       that such additional attribution notices cannot be construed
*       as modifying the License.
*
*     You may add Your own copyright statement to Your modifications and
*     may provide additional or different license terms and conditions
*     for use, reproduction, or distribution of Your modifications, or
*     for any such Derivative Works as a whole, provided Your use,
*     reproduction, and distribution of the Work otherwise complies with
*     the conditions stated in this License.
*
*   5. Submission of Contributions. Unless You explicitly state otherwise,
*     any Contribution intentionally submitted for inclusion in the Work
*     by You to the Licensor shall be under the terms and conditions of
*     this License, without any additional terms or conditions.
*     Notwithstanding the above, nothing herein shall supersede or modify
*     the terms of any separate license agreement you may have executed
*     with Licensor regarding such Contributions.
*
*   6. Trademarks. This License does not grant permission to use the trade
*     names, trademarks, service marks, or product names of the Licensor,
*     except as required for reasonable and customary use in describing the
*     origin of the Work and reproducing the content of the NOTICE file.
*
*   7. Disclaimer of Warranty. Unless required by applicable law or
*     agreed to in writing, Licensor provides the Work (and each
*     Contributor provides its Contributions) on an "AS IS" BASIS,
*     WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
*     implied, including, without limitation, any warranties or conditions
*     of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
*     PARTICULAR PURPOSE. You are solely responsible for determining the
*     appropriateness of using or redistributing the Work and assume any
*     risks associated with Your exercise of permissions under this License.
*
*   8. Limitation of Liability. In no event and under no legal theory,
*     whether in tort (including negligence), contract, or otherwise,
*     unless required by applicable law (such as deliberate and grossly
*     negligent acts) or agreed to in writing, shall any Contributor be
*     liable to You for damages, including any direct, indirect, special,
*     incidental, or consequential damages of any character arising as a
```

```
*     result of this License or out of the use or inability to use the
*     Work (including but not limited to damages for loss of goodwill,
*     work stoppage, computer failure or malfunction, or any and all
*     other commercial damages or losses), even if such Contributor
*     has been advised of the possibility of such damages.
*
*   9. Accepting Warranty or Additional Liability. While redistributing
*     the Work or Derivative Works thereof, You may choose to offer,
*     and charge a fee for, acceptance of support, warranty, indemnity,
*     or other liability obligations and/or rights consistent with this
*     License. However, in accepting such obligations, You may act only
*     on Your own behalf and on Your sole responsibility, not on behalf
*     of any other Contributor, and only if You agree to indemnify,
*     defend, and hold each Contributor harmless for any liability
*     incurred by, or claims asserted against, such Contributor by reason
*     of your accepting any such warranty or additional liability.
*
*   END OF TERMS AND CONDITIONS
*
*   APPENDIX: How to apply the Apache License to your work.
*
*     To apply the Apache License to your work, attach the following
*     boilerplate notice, with the fields enclosed by brackets "[]"
*     replaced with your own identifying information. (Don't include
*     the brackets!)  The text should be enclosed in the appropriate
*     comment syntax for the file format. We also recommend that a
*     file or class name and description of purpose be included on the
*     same "printed page" as the copyright notice for easier
*     identification within third-party archives.
*
*   Copyright [yyyy] [name of copyright owner]
*
*   Licensed under the Apache License, Version 2.0 (the "License");
*   you may not use this file except in compliance with the License.
*   You may obtain a copy of the License at
*
*       http://www.apache.org/licenses/LICENSE-2.0
*
*   Unless required by applicable law or agreed to in writing, software
*   distributed under the License is distributed on an "AS IS" BASIS,
*   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
*   See the License for the specific language governing permissions and
*   limitations under the License.
*/
```

This license came from:
http://www.w3.org/Consortium/Legal/copyright-software-19980720

W3C SOFTWARE NOTICE AND LICENSE
Copyright (c) 1994-2001 World
Wide Web Consortium, World
Wide Web Consortium, (<a href=
"http://www.lcs.mit.edu/">Massachusetts Institute of
Technology, Institut National de
Recherche en Informatique et en Automatique, <a href=
"http://www.keio.ac.jp/">Keio University). All Rights Reserved.
http://www.w3.org/Consortium/Legal/

This W3C work (including software, documents, or other related
items) is being provided by the copyright holders under the
following license. By obtaining, using and/or copying this work,
you (the licensee) agree that you have read, understood, and will
comply with the following terms and conditions:
Permission to use, copy, modify, and distribute this software
and its documentation, with or without modification, for any
purpose and without fee or royalty is hereby granted, provided that
you include the following on ALL copies of the software and
documentation or portions thereof, including modifications, that
you make:

The full text of this NOTICE in a location viewable to users of
the redistributed or derivative work.

Any pre-existing intellectual property disclaimers, notices, or
terms and conditions. If none exist, a short notice of the
following form (hypertext is preferred, text is permitted) should
be used within the body of any redistributed or derivative code:
"Copyright (c) [$date-of-software] World Wide Web Consortium, (Massachusetts Institute of
Technology, Institut National de
Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.
http://www.w3.org/Consortium/Legal/"

Notice of any changes or modifications to the W3C files,
including the date changes were made. (We recommend you provide
URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND
COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE
USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD
PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT,
SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE
SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in
advertising or publicity pertaining to the software without
specific, written prior permission. Title to copyright in this
software and any associated documentation will at all times remain
with copyright holders.

This formulation of W3C's notice and license became active on
August 14 1998 so as to improve compatibility with GPL. This
version ensures that W3C software licensing terms are no more
restrictive than GPL and consequently W3C software may be
distributed in GPL packages. See the older formulation for the
policy prior to this date. Please see our Copyright FAQ for common
questions about using materials from
our site, including specific terms and conditions for packages like
libwww, Amaya, and Jigsaw.
Other questions about this notice can be
directed to site-policy@w3.org.

webmaster
This license came from: http://www.megginson.com/SAX/copying.html
However please note future versions of SAX may be covered
under http://saxproject.org/?selected=pd

This page is now out of date -- see the new SAX site at
http://www.saxproject.org/ for more up-to-date
releases and other information. Please change your bookmarks.

SAX2 is Free!

I hereby abandon any property rights to SAX 2.0 (the Simple API for
XML), and release all of the SAX 2.0 source code, compiled code, and
documentation contained in this distribution into the Public Domain.
SAX comes with NO WARRANTY or guarantee of fitness for any
purpose.

David Megginson, david@megginson.com
2000-05-05
/*
* The Apache Software License, Version 1.1
*
*
* Copyright (c) 1999-2002 The Apache Software Foundation. All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions

```
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. The end-user documentation included with the redistribution,
*    if any, must include the following acknowledgment:
*       "This product includes software developed by the
*        Apache Software Foundation (http://www.apache.org/)."
*    Alternately, this acknowledgment may appear in the software itself,
*    if and wherever such third-party acknowledgments normally appear.
*
* 4. The names "Xerces" and "Apache Software Foundation" must
*    not be used to endorse or promote products derived from this
*    software without prior written permission. For written
*    permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache",
*    nor may "Apache" appear in their name, without prior written
*    permission of the Apache Software Foundation.
*
* THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* ====================================================================
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation and was
* originally based on software copyright (c) 1999, International
* Business Machines, Inc., http://www.ibm.com.  For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*/
```

1.5 asm-all-3.3.1_within-cglib 3.3.1

1.5.1 Available under license :

Copyright (c) 2000-2005 INRIA, France Telecom
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holders nor the names of its
   contributors may be used to endorse or promote products derived from
   this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.

1.6 aspectjrt 1.6.11

1.6.1 Available under license :

Eclipse Public License - v 1.0
THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided

that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or

offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable

separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

1.7 aspectjweaver 1.6.11

1.7.1 Available under license :

Eclipse Public License - v 1.0
THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

1.8 axiom-api-1.2.4 1.2.4

1.8.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of

the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works

that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A

PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.9 axiom-impl-1.2.4 1.2.4

1.9.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate

as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify

the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include

the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.10 axis.jar 1.4

1.10.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed

as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this

License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).
===
== NOTICE file corresponding to section 4(d) of the Apache License, ==
== Version 2.0, in this case for the Apache Axis distribution. ==
===

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.11 axis2-adb-1.2 1.2

1.11.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity
on behalf of whom a Contribution has been received by Licensor and
subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
copyright license to reproduce, prepare Derivative Works of,
publicly display, publicly perform, sublicense, and distribute the
Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
(except as stated in this section) patent license to make, have made,
use, offer to sell, sell, import, and otherwise transfer the Work,
where such license applies only to those patent claims licensable
by such Contributor that are necessarily infringed by their
Contribution(s) alone or by combination of their Contribution(s)
with the Work to which such Contribution(s) was submitted. If You
institute patent litigation against any entity (including a
cross-claim or counterclaim in a lawsuit) alleging that the Work
or a Contribution incorporated within the Work constitutes direct
or contributory patent infringement, then any patent licenses
granted to You under this License for that Work shall terminate
as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or

Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

```
=========================================================================
==  NOTICE file corresponding to the section 4 d of                    ==
==  the Apache License, Version 2.0,                                   ==
==  in this case for the Apache Axis2 distribution.                    ==
=========================================================================
```

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).
Portions Copyright 2006 International Business Machines Corp.
Portions Copyright 2005-2007 WSO2, Inc.

This product also includes schemas and specification developed by:
- the W3C consortium (http://www.w3c.org)

This product also includes WS-* schemas developed by International
Business Machines Corporation, Microsoft Corporation, BEA Systems,
TIBCO Software, SAP AG, Sonic Software, and VeriSign

This product also includes a WSDL developed by salesforce.com
- Copyright 1999-2006 salesforce.com, inc.

Portions of the included xmlbeans library were originally based on the following:
- software copyright (c) 2000-2003, BEA Systems, <http://www.bea.com/>.

Please read the different LICENSE files present in the lib directory of
this distribution.

1.12 axis2-kernel 1.2

1.12.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent

to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work,

excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any

risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0 1.

Definitions.

1.1. Contributor means each individual or entity that creates or
contributes to the creation of Modifications.

1.2. Contributor Version means the combination of the Original
Software, prior Modifications used by a Contributor (if any), and
the Modifications made by that particular Contributor.

1.3. Covered Software means (a) the Original Software, or (b)
Modifications, or (c) the combination of files containing
Original Software with files containing Modifications, in each
case including portions thereof.

1.4. Executable means the Covered Software in any form other than
Source Code.

1.5. Initial Developer means the individual or entity that first makes
Original Software available under this License.

1.6. Larger Work means a work which combines Covered Software or
portions thereof with code not governed by the terms of this
License.

1.7. License means this document.

1.8. Licensable means having the right to grant, to the maximum extent
possible, whether at the time of the initial grant or
subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications means the Source Code and Executable form of any of
the following: A. Any file that results from an addition to,
deletion from or modification of the contents of a file
containing Original Software or previous Modifications; B. Any
new file that contains any part of the Original Software or
previous Modification; or C. Any new file that is contributed or
otherwise made available under the terms of this License.

1.10. Original Software means the Source Code and Executable form of
computer software code that is originally released under this
License.

1.11. Patent Claims means any patent claim(s), now owned or hereafter

acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. Source Code means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. You (or Your) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, You includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, control means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant. Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof);

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License;

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant. Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property

claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code. Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications. The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original

creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices. You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms. You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions. You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipients rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works. You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions. Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions. You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions. When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY. COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN AS IS BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the

breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as Participant) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY. UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTYS NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS. The Covered Software is a commercial item, as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of commercial computer software (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and commercial computer software documentation as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48

C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS. This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdictions conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS. As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) The code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.
Copyright (c) 2000 - 2006 The Legion Of The Bouncy Castle (http://www.bouncycastle.org)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the

rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
The software comprising backport-util-concurrent is based in large
part on the code from JSR166, and the package dl.util.concurrent.
The software has been released to the public domain, as explained at:
http://creativecommons.org/licenses/publicdomain, excepting portions
of the class
edu.emory.mathcs.backport.java.util.concurrent.CopyOnWriteArrayList,
which were adapted from class java.util.ArrayList, written by Sun
Microsystems, Inc, which are used with kind permission, and subject
to the following:

Copyright 2002-2004 Sun Microsystems, Inc. All rights reserved. Use is
subject to the following license terms.

"Sun hereby grants you a non-exclusive, worldwide, non-transferrable
license to use and distribute the Java Software technologies as part
of a larger work in source and binary forms, with or without
modification, provided that the following conditions are met:

-Neither the name of or trademarks of Sun may be used to endorse or
promote products derived from the Java Software technology without
specific prior written permission.

-Redistributions of source or binary code must be accompanied by the
following notice and disclaimers:

Portions copyright Sun Microsystems, Inc. Used with kind permission.

This software is provided AS IS, without a warranty of any kind. ALL
EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND
WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PUPOSE OR
NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN
MICROSYSTEMS, INC. AND ITS LICENSORS SHALL NOT BE LIABLE
FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF
USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS
DERIVATIVES. IN NO EVENT WILL SUN MICROSYSTEMS, INC. OR

ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT,CONSQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OR LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN MICROSYSTEMS, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility."

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the listed copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.
* Neither the name of JiBX nor the names of its contributors may be used
  to endorse or promote products derived from this software without specific
  prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Copyright 2003 (C) James Strachan and Bob Mcwhirter. All Rights Reserved.

Redistribution and use of this software and associated documentation
("Software"), with or without modification, are permitted provided
that the following conditions are met:

1. Redistributions of source code must retain copyright
   statements and notices. Redistributions must also contain a
   copy of this document.

2. Redistributions in binary form must reproduce the
   above copyright notice, this list of conditions and the
   following disclaimer in the documentation and/or other
   materials provided with the distribution.

3. The name "groovy" must not be used to endorse or promote
   products derived from this Software without prior written
   permission of The Codehaus. For written permission,
   please contact info@codehaus.org.

4. Products derived from this Software may not be called "groovy"
   nor may "groovy" appear in their names without prior written
   permission of The Codehaus. "groovy" is a registered
   trademark of The Codehaus.

5. Due credit should be given to The Codehaus -
   http://groovy.codehaus.org/

THIS SOFTWARE IS PROVIDED BY THE CODEHAUS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
THE CODEHAUS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
===
== NOTICE file corresponding to the section 4 d of ==
== the Apache License, Version 2.0, ==
== in this case for the Apache Axis2 distribution. ==
===

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).
Portions Copyright 2006 International Business Machines Corp.

Please read the different LICENSE files present in the lib directory of
this distribution.
Common Public License Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in

conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient

copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is

intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY

WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

1.13 batik-ext 1.7

1.13.1 Available under license :

xml-commons/java/external/LICENSE.dom-documentation.txt $Id: LICENSE.dom-documentation.txt 201084 2002-12-09 16:15:21Z vhardy $

This license came from: http://www.w3.org/Consortium/Legal/copyright-documents-19990405

W3C DOCUMENT NOTICE AND LICENSE
Copyright 1994-2001 World
Wide Web Consortium, World
Wide Web Consortium, (<a href=
"http://www.lcs.mit.edu/">Massachusetts Institute of
Technology, Institut National de
Recherche en Informatique et en Automatique, <a href=
"http://www.keio.ac.jp/">Keio University). All Rights Reserved.
http://www.w3.org/Consortium/Legal/

Public documents on the W3C site are provided by the copyright holders under the following license. The software or Document Type Definitions (DTDs) associated with W3C specifications are governed by the Software Notice. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

A link or URL to the original W3C document.

The pre-existing copyright notice of the original author, or if it doesn't exist, a notice of the form: "Copyright [$date-of-document] World Wide Web Consortium, (Massachusetts
Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio

University). All Rights Reserved.
http://www.w3.org/Consortium/Legal/" (Hypertext is preferred, but a
textual representation is permitted.)

If it exists, the STATUS of the W3C document.

When space permits, inclusion of the full text of this NOTICE
should be provided. We request that authorship
attribution be provided in any software, documents, or other items
or products that you create pursuant to the implementation of the
contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents
is granted pursuant to this license. However, if additional
requirements (documented in the Copyright
FAQ) are satisfied, the right to create modifications or
derivatives issometimes granted by the W3C to individuals
complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO
REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS
OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE
IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY
PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT,
SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE
DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS
THEREOF.

The name and trademarks of copyright holders may NOT be used in
advertising or publicity pertaining to this document or its
contents without specific, written prior permission. Title to
copyright in this document will at all times remain with copyright
holders.

This formulation of W3C's notice and license became active on
April 05 1999 so as to account for the treatment of DTDs, schema's and
bindings. See the older formulation for the policy prior to this date.
Please see
our Copyright FAQ for common questions
about using materials from our site, including specific terms and
conditions for packages like libwww, Amaya, and Jigsaw.
Other questions about this notice can be directed to site-policy@w3.org.

webmaster

(last updated by reagle on 1999/04/99.)
xml-commons/java/external/LICENSE.dom-software.txt $Id: LICENSE.dom-software.txt 201084 2002-12-09 16:15:21Z vhardy $

This license came from: http://www.w3.org/Consortium/Legal/copyright-software-19980720

W3C SOFTWARE NOTICE AND LICENSE
Copyright 1994-2001 World
Wide Web Consortium, World
Wide Web Consortium, (<a href=
"http://www.lcs.mit.edu/">Massachusetts Institute of
Technology, Institut National de
Recherche en Informatique et en Automatique, <a href=
"http://www.keio.ac.jp/">Keio University). All Rights Reserved.
http://www.w3.org/Consortium/Legal/

This W3C work (including software, documents, or other related
items) is being provided by the copyright holders under the
following license. By obtaining, using and/or copying this work,
you (the licensee) agree that you have read, understood, and will
comply with the following terms and conditions:
Permission to use, copy, modify, and distribute this software
and its documentation, with or without modification, for any
purpose and without fee or royalty is hereby granted, provided that
you include the following on ALL copies of the software and
documentation or portions thereof, including modifications, that
you make:

The full text of this NOTICE in a location viewable to users of
the redistributed or derivative work.

Any pre-existing intellectual property disclaimers, notices, or
terms and conditions. If none exist, a short notice of the
following form (hypertext is preferred, text is permitted) should
be used within the body of any redistributed or derivative code:
"Copyright [$date-of-software] World Wide Web Consortium, (Massachusetts Institute of
Technology, Institut National de
Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.
http://www.w3.org/Consortium/Legal/"

Notice of any changes or modifications to the W3C files,
including the date changes were made. (We recommend you provide
URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND
COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE
USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD
PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT,
SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE
SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in
advertising or publicity pertaining to the software without
specific, written prior permission. Title to copyright in this
software and any associated documentation will at all times remain
with copyright holders.

This formulation of W3C's notice and license became active on
August 14 1998 so as to improve compatibility with GPL. This
version ensures that W3C software licensing terms are no more
restrictive than GPL and consequently W3C software may be
distributed in GPL packages. See the older formulation for the
policy prior to this date. Please see our Copyright FAQ for common
questions about using materials from
our site, including specific terms and conditions for packages like
libwww, Amaya, and Jigsaw.
Other questions about this notice can be
directed to site-policy@w3.org.

webmaster
(last updated $Date: 2002-12-09 17:15:21 +0100 (Mon, 09 Dec 2002) $)

1.14 batik-util 1.7

1.14.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all

other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and

subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed

as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the

Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache Batik
Copyright 1999-2007 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

This software contains code from the World Wide Web Consortium (W3C) for the Document Object Model API (DOM API) and SVG Document Type Definition (DTD).

This software contains code from the International Organisation for Standardization for the definition of character entities used in the software's documentation.

This product includes images from the Tango Desktop Project (http://tango.freedesktop.org/).

This product includes images from the Pasodoble Icon Theme (http://www.jesusda.com/projects/pasodoble).

1.15 cglib 2.2.2

1.15.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but

not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity
on behalf of whom a Contribution has been received by Licensor and
subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
copyright license to reproduce, prepare Derivative Works of,
publicly display, publicly perform, sublicense, and distribute the
Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
(except as stated in this section) patent license to make, have made,
use, offer to sell, sell, import, and otherwise transfer the Work,
where such license applies only to those patent claims licensable
by such Contributor that are necessarily infringed by their

Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with

the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

ASM: a very small and fast Java bytecode manipulation framework
Copyright (c) 2000,2002,2003 INRIA, France Telecom
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holders nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE

LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.16 com.cisco.xmp.osgi.slf4j-api 1.5.8.PATCHED

1.16.1 Available under license :

Copyright (c) 2004-2008 QOS.ch All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.17 com.springsource.com.mysql.jdbc 5.1.6

1.17.1 Available under license :

/*
Copyright (C) 2002-2004 MySQL AB

This program is free software; you can redistribute it and/or modify
it under the terms of version 2 of the GNU General Public License as
published by the Free Software Foundation.

There are special exceptions to the terms and conditions of the GPL
as it is applied to this software. View the full text of the
exception in file EXCEPTIONS-CONNECTOR-J in the directory of this
software distribution.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

*/
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users. This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it. (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.) You can apply it to
your programs, too.

When we speak of free software, we are referring to freedom, not
price. Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have. You must make sure that they, too, receive or can get the
source code. And you must show them these terms so they know their
rights.

We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,

distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the

Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding

those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs. If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library. If this is what you want to do, use the GNU Lesser General
Public License instead of this License.

1.18 com.springsource.javax.activation 1.1.1

1.18.1 Available under license :

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0 1.

Definitions.

1.1. Contributor means each individual or entity that creates or
contributes to the creation of Modifications.

1.2. Contributor Version means the combination of the Original
Software, prior Modifications used by a Contributor (if any), and
the Modifications made by that particular Contributor.

1.3. Covered Software means (a) the Original Software, or (b)
Modifications, or (c) the combination of files containing
Original Software with files containing Modifications, in each
case including portions thereof.

1.4. Executable means the Covered Software in any form other than
Source Code.

1.5. Initial Developer means the individual or entity that first makes
Original Software available under this License.

1.6. Larger Work means a work which combines Covered Software or
portions thereof with code not governed by the terms of this
License.

1.7. License means this document.

1.8. Licensable means having the right to grant, to the maximum extent
possible, whether at the time of the initial grant or
subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications means the Source Code and Executable form of any of the following: A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications; B. Any new file that contains any part of the Original Software or previous Modification; or C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. Original Software means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. Patent Claims means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. Source Code means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. You (or Your) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, You includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, control means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant. Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof);

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License;

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant. Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code. Any Covered Software that You

distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications. The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices. You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms. You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions. You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipients rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License

are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works. You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions. Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions. You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions. When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY. COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN AS IS BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE

QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as Participant) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY. UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE

BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTYS NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS. The Covered Software is a commercial item, as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of commercial computer software (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and commercial computer software documentation as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS. This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdictions conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS. As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute

any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) The code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and

(2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and

you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of

Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by

modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License

may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded. In such case, this License incorporates
the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time. Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number. If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation. If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission. For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this. Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
`Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs. If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library. If this is what you want to do, use the GNU Lesser General
Public License instead of this License.

1.19 com.springsource.org.antlr 3.0.1

1.19.1 Available under license :

/*
[The "BSD licence"]
Copyright (c) 2005-2006 Terence Parr
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products
   derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS" AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

1.20 com.springsource.org.antlr.stringtemplate-3.1.0.b1.jar 3.1.0.b1

1.20.1 Available under license :

Copyright (c) 2003-2007, Terence Parr
All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.21 com.springsource.org.apache.bcel 5.1.0

1.21.1 Notifications :

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

1.21.2 Available under license :

/* ==
* The Apache Software License, Version 1.1
*
* Copyright (c) 2001 The Apache Software Foundation. All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*

```
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. The end-user documentation included with the redistribution,
*    if any, must include the following acknowledgment:
*       "This product includes software developed by the
*        Apache Software Foundation (http://www.apache.org/)."
*    Alternately, this acknowledgment may appear in the software itself,
*    if and wherever such third-party acknowledgments normally appear.
*
* 4. The names "Apache" and "Apache Software Foundation" and
*    "Apache BCEL" must not be used to endorse or promote products
*    derived from this software without prior written permission. For
*    written permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache",
*    "Apache BCEL", nor may "Apache" appear in their name, without
*    prior written permission of the Apache Software Foundation.
*
* THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* ====================================================================
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation.  For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*/
```

1.22 com.springsource.org.apache.commons.cli 1.1.0

1.22.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed

with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate

comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache Commons CLI
Copyright 2001-2007 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.23 com.springsource.org.apache.commons.codec 1.4.0

1.23.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all

other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and

subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed

as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the

Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache Commons Codec
Copyright 2002-2009 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

---

src/test/org/apache/commons/codec/language/DoubleMetaphoneTest.java contains
test data from http://aspell.sourceforge.net/test/batch0.tab.

Copyright (C) 2002 Kevin Atkinson (kevina@gnu.org). Verbatim copying
and distribution of this entire article is permitted in any medium,
provided this notice is preserved.

1.24 com.springsource.org.apache.commons.collections 3.2.1

1.25 com.springsource.org.apache.commons.collections 3.2.0

1.25.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.
You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.
5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend,

and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.26 com.springsource.org.apache.commons.fileupload 1.2.1

1.26.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but

excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its

distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise,

unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache Commons FileUpload
Copyright 2002-2008 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.27 com.springsource.org.apache.commons.httpclient 3.1.0

1.27.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You

institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.28 com.springsource.org.apache.commons.io 1.4.0

1.28.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity

on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one

of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a

result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache Commons IO
Copyright 2001-2008 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.29 com.springsource.org.apache.commons.lang 2.4.0

1.29.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate

as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify

the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include

the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache Commons Lang
Copyright 2001-2008 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.30 com.springsource.org.apache.commons.logging 1.1.1

1.30.1 Available under license :

```
/*
* Licensed to the Apache Software Foundation (ASF) under one or more
* contributor license agreements.  See the NOTICE file distributed with
* this work for additional information regarding copyright ownership.
* The ASF licenses this file to You under the Apache License, Version 2.0
* (the "License"); you may not use this file except in compliance with
* the License.  You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
```

*

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity
on behalf of whom a Contribution has been received by Licensor and
subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
copyright license to reproduce, prepare Derivative Works of,
publicly display, publicly perform, sublicense, and distribute the
Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
(except as stated in this section) patent license to make, have made,
use, offer to sell, sell, import, and otherwise transfer the Work,
where such license applies only to those patent claims licensable
by such Contributor that are necessarily infringed by their
Contribution(s) alone or by combination of their Contribution(s)
with the Work to which such Contribution(s) was submitted. If You
institute patent litigation against any entity (including a
cross-claim or counterclaim in a lawsuit) alleging that the Work
or a Contribution incorporated within the Work constitutes direct
or contributory patent infringement, then any patent licenses
granted to You under this License for that Work shall terminate
as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or

agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.31 com.springsource.org.apache.commons.net 2.0.0

1.31.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made,

use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions

for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability

incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache Commons Net
Copyright 2001-2008 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.32 com.springsource.org.apache.log4j 1.2.15

1.32.1 Available under license :

/*
* Licensed to the Apache Software Foundation (ASF) under one or more
* contributor license agreements. See the NOTICE file distributed with
* this work for additional information regarding copyright ownership.
* The ASF licenses this file to You under the Apache License, Version 2.0
* (the "License"); you may not use this file except in compliance with

* the License. You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate

as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify

the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include

the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.33 com.springsource.org.apache.oro 2.0.8

1.33.1 Notifications :

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

1.33.2 Available under license :

* ==
* The Apache Software License, Version 1.1
*
* Copyright (c) 2000 The Apache Software Foundation. All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*   notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*   notice, this list of conditions and the following disclaimer in
*   the documentation and/or other materials provided with the
*   distribution.
*
* 3. The end-user documentation included with the redistribution,
*   if any, must include the following acknowledgment:
*   "This product includes software developed by the

* Apache Software Foundation (http://www.apache.org/)."
* Alternately, this acknowledgment may appear in the software itself,
* if and wherever such third-party acknowledgments normally appear.
*
* 4. The names "Apache" and "Apache Software Foundation", "Jakarta-Oro"
* must not be used to endorse or promote products derived from this
* software without prior written permission. For written
* permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache"
* or "Jakarta-Oro", nor may "Apache" or "Jakarta-Oro" appear in their
* name, without prior written permission of the Apache Software Foundation.
*
* THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* ==

1.34 com.springsource.org.apache.regexp 1.5.0

1.34.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity

on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one

of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a

result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.35 com.springsource.org.apache.velocity 1.6.4

1.35.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the

editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the

Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the

same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache Velocity

Copyright (C) 2000-2007 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

1.36 com.springsource.org.apache.xalan 2.7.1

1.36.1 Available under license :

```
 ========================================================================
 ==  NOTICE file corresponding to section 4(d) of the Apache License,   ==
 ==  Version 2.0, in this case for the Apache xml-commons xml-apis      ==
 ==  distribution.                                                      ==
 ========================================================================
```

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

Portions of this software were originally based on the following:
- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
- software copyright (c) 2000 World Wide Web Consortium, http://www.w3.org
xml-commons/java/external/LICENSE.dom-software.txt $Id: LICENSE.dom-software.txt,v 1.2 2005/06/03 22:49:13 mrglavas Exp $

This license came from: http://www.w3.org/Consortium/Legal/copyright-software-20021231

W3C SOFTWARE NOTICE AND LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.
3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002. This version removes the copyright ownership notice such that this license can be used with materials other than those owned by the W3C, reflects that ERCIM is now a host of the W3C, includes references to this specific dated version of the license, and removes the ambiguous grant of "use". Otherwise, this version is the same as the previous version and is written so as to preserve the Free

Software Foundation's assessment of GPL compatibility and OSI's certification under the Open Source Definition. Please see our Copyright FAQ for common questions about using materials from our site, including specific terms and conditions for packages like libwww, Amaya, and Jigsaw. Other questions about this notice can be directed to site-policy@w3.org.

Joseph Reagle <site-policy@w3.org>

Last revised by Reagle $Date: 2005/06/03 22:49:13 $
xml-commons/java/external/LICENSE.sax.txt $Id: LICENSE.sax.txt,v 1.1 2002/01/31 23:26:48 curcuru Exp $

This license came from: http://www.megginson.com/SAX/copying.html
However please note future versions of SAX may be covered
under http://saxproject.org/?selected=pd

This page is now out of date -- see the new SAX site at
http://www.saxproject.org/ for more up-to-date
releases and other information. Please change your bookmarks.

SAX2 is Free!

I hereby abandon any property rights to SAX 2.0 (the Simple API for
XML), and release all of the SAX 2.0 source code, compiled code, and
documentation contained in this distribution into the Public Domain.
SAX comes with NO WARRANTY or guarantee of fitness for any
purpose.

David Megginson, david@megginson.com
2000-05-05
xml-commons/java/external/LICENSE.dom-documentation.txt $Id: LICENSE.dom-documentation.txt,v 1.2
2005/06/03 22:49:13 mrglavas Exp $

This license came from: http://www.w3.org/Consortium/Legal/copyright-documents-20021231

W3C DOCUMENT LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

Public documents on the W3C site are provided by the copyright holders under the following license. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231"
3. If it exists, the STATUS of the W3C document.

When space permits, inclusion of the full text of this NOTICE should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002. This version removes the copyright ownership notice such that this license can be used with materials other than those owned by the W3C, moves information on style sheets, DTDs, and schemas to the Copyright FAQ, reflects that ERCIM is

now a host of the W3C, includes references to this specific dated version of the license, and removes the ambiguous grant of "use". See the older formulation for the policy prior to this date. Please see our Copyright FAQ for common questions about using materials from our site, such as the translating or annotating specifications. Other questions about this notice can be directed to site-policy@w3.org.

Joseph Reagle <site-policy@w3.org>

Last revised by Reagle $Date: 2005/06/03 22:49:13 $

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work

(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses

granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]"

replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.37 com.springsource.org.apache.xerces 2.9.1

1.37.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the

outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity
on behalf of whom a Contribution has been received by Licensor and
subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable

copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and

do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!) The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.38 com.springsource.org.apache.xml.resolver 1.2.0

1.38.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity
on behalf of whom a Contribution has been received by Licensor and
subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
copyright license to reproduce, prepare Derivative Works of,
publicly display, publicly perform, sublicense, and distribute the
Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
(except as stated in this section) patent license to make, have made,
use, offer to sell, sell, import, and otherwise transfer the Work,
where such license applies only to those patent claims licensable
by such Contributor that are necessarily infringed by their
Contribution(s) alone or by combination of their Contribution(s)
with the Work to which such Contribution(s) was submitted. If You
institute patent litigation against any entity (including a
cross-claim or counterclaim in a lawsuit) alleging that the Work
or a Contribution incorporated within the Work constitutes direct
or contributory patent infringement, then any patent licenses
granted to You under this License for that Work shall terminate
as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or

Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache XML Commons Resolver
Copyright 2006 The Apache Software Foundation.

This product includes software developed at
The Apache Software Foundation http://www.apache.org/

Portions of this code are derived from classes placed in the
public domain by Arbortext on 10 Apr 2000. See:
http://www.arbortext.com/customer_support/updates_and_technical_notes/catalogs/docs/README.htm

1.39 com.springsource.org.apache.xml.serializer 2.7.1

1.39.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the

direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of

this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and

wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor

has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.40 com.springsource.org.apache.xmlcommons 1.3.4

1.40.1 Available under license :

IP Central Maven plugin could not find license data for this component. Please use the source code or component links to investigate and identify the correct license text to insert here.

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain

separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the

origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
xml-commons/java/external/LICENSE.dom-documentation.txt $Id: LICENSE.dom-documentation.txt 226215 2005-06-03 22:49:13Z mrglavas $

This license came from: http://www.w3.org/Consortium/Legal/copyright-documents-20021231

W3C DOCUMENT LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

Public documents on the W3C site are provided by the copyright holders under the following license. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231"
3. If it exists, the STATUS of the W3C document.

When space permits, inclusion of the full text of this NOTICE should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002. This version removes the copyright ownership notice such that this license can be used with materials other than those owned by the W3C, moves information on style sheets, DTDs, and schemas to the Copyright FAQ, reflects that ERCIM is now a host of the W3C, includes references to this specific dated version of the license, and removes the ambiguous grant of "use". See the older formulation for the policy prior to this date. Please see our Copyright FAQ for common questions about using materials from our site, such as the translating or annotating specifications. Other questions about this notice can be directed to site-policy@w3.org.

Joseph Reagle <site-policy@w3.org>

Last revised by Reagle $Date: 2005-06-03 18:49:13 -0400 (Fri, 03 Jun 2005) $ xml-commons/java/external/LICENSE.dom-software.txt $Id: LICENSE.dom-software.txt 226215 2005-06-03 22:49:13Z mrglavas $

This license came from: http://www.w3.org/Consortium/Legal/copyright-software-20021231

W3C SOFTWARE NOTICE AND LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.
3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002. This version removes the copyright ownership notice such that this license can be used with materials other than those owned by the W3C, reflects that ERCIM is now a host of the W3C, includes references to this specific dated version of the license, and removes the ambiguous grant of "use". Otherwise, this version is the same as the previous version and is written so as to preserve the Free Software Foundation's assessment of GPL compatibility and OSI's certification under the Open Source Definition. Please see our Copyright FAQ for common questions about using materials from our site, including specific terms and conditions for packages like libwww, Amaya, and Jigsaw. Other questions about

this notice can be directed to site-policy@w3.org.

Joseph Reagle <site-policy@w3.org>

Last revised by Reagle $Date: 2005-06-03 18:49:13 -0400 (Fri, 03 Jun 2005) $
xml-commons/java/external/LICENSE.sax.txt $Id: LICENSE.sax.txt 225954 2002-01-31 23:26:48Z curcuru $

This license came from: http://www.megginson.com/SAX/copying.html
However please note future versions of SAX may be covered
under http://saxproject.org/?selected=pd

This page is now out of date -- see the new SAX site at
http://www.saxproject.org/ for more up-to-date
releases and other information. Please change your bookmarks.

SAX2 is Free!

I hereby abandon any property rights to SAX 2.0 (the Simple API for
XML), and release all of the SAX 2.0 source code, compiled code, and
documentation contained in this distribution into the Public Domain.
SAX comes with NO WARRANTY or guarantee of fitness for any
purpose.

David Megginson, david@megginson.com
2000-05-05

```
========================================================================
 ==  NOTICE file corresponding to section 4(d) of the Apache License,   ==
 ==  Version 2.0, in this case for the Apache xml-commons xml-apis      ==
 ==  distribution.                                                     ==
 ========================================================================

 Apache XML Commons XML APIs
 Copyright 2006 The Apache Software Foundation.

 This product includes software developed at
 The Apache Software Foundation (http://www.apache.org/).

 Portions of this software were originally based on the following:
   - software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
   - software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
   - software copyright (c) 2000 World Wide Web Consortium, http://www.w3.org
```

1.41 com.springsource.org.custommonkey.xmlunit-1.2.0.jar 1.2.0

1.41.1 Available under license :

1.42 com.springsource.org.mozilla.javascript 1.7.0.R1

1.42.1 Available under license :

MOZILLA PUBLIC LICENSE

Version 1.1

---

1. Definitions.

1.0.1. "Commercial Use" means distribution or otherwise making the Covered Code available to a third party.

1.1. "Contributor" means each entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. "Covered Code" means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof.

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. "License" means this document.

1.8.1. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. "Original Code" means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.10.1. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.11. "Source Code" means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated interface definition files, scripts used to control compilation and installation of an Executable, or source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor's choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. Source Code License.

2.1. The Initial Developer Grant.
The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce,

modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work; and

(b) under Patents Claims infringed by the making, using or selling of Original Code, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Code (or portions thereof).

(c) the licenses granted in this Section 2.1(a) and (b) are effective on the date Initial Developer first distributes Original Code under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: 1) for code that You delete from the Original Code; 2) separate from the Original Code; or 3) for infringements caused by: i) the modification of the Original Code or ii) the combination of the Original Code with other software or devices.

2.2. Contributor Grant.
Subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor, to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: 1) Modifications made by that Contributor (or portions thereof); and 2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) the licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first makes Commercial Use of the Covered Code.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: 1) for any code that Contributor has deleted from the Contributor Version; 2) separate from the Contributor Version; 3) for infringements caused by: i) third party modifications of Contributor Version or ii) the combination of Modifications made

by that Contributor with other software (except as part of the Contributor Version) or other devices; or 4) under Patent Claims infringed by Covered Code in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Application of License.
The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients' rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

3.2. Availability of Source Code.
Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.

3.3. Description of Modifications.
You must cause all Covered Code to which You contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

3.4. Intellectual Property Matters
(a) Third Party Claims.
If Contributor has knowledge that a license under a third party's intellectual property rights is required to exercise the rights granted by such Contributor under Sections 2.1 or 2.2,

Contributor must include a text file with the Source Code distribution titled "LEGAL" which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If Contributor obtains such knowledge after the Modification is made available as described in Section 3.2, Contributor shall promptly modify the LEGAL file in all copies Contributor makes available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs.
If Contributor's Modifications include an application programming interface and Contributor has knowledge of patent licenses which are reasonably necessary to implement that API, Contributor must also include this information in the LEGAL file.

(c) Representations.
Contributor represents that, except as disclosed pursuant to Section 3.4(a) above, Contributor believes that Contributor's Modifications are Contributor's original creation(s) and/or Contributor has sufficient rights to grant the rights conveyed by this License.

3.5. Required Notices.
You must duplicate the notice in Exhibit A in each file of the Source Code. If it is not possible to put such notice in a particular Source Code file due to its structure, then You must include such notice in a location (such as a relevant directory) where a user would be likely to look for such a notice. If You created one or more Modification(s) You may add your name as a Contributor to the notice described in Exhibit A. You must also duplicate this License in any documentation for the Source Code where You describe recipients' rights or ownership rights relating to Covered Code. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. Distribution of Executable Versions.
You may distribute Covered Code in Executable form only if the requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of

the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code or ownership rights under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.7. Larger Works.
You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Application of this License.

This License applies to code to which the Initial Developer has attached the notice in Exhibit A and to related Covered Code.

6. Versions of the License.

6.1. New Versions.
Netscape Communications Corporation ("Netscape") may publish revised and/or new versions of the License from time to time. Each version

will be given a distinguishing version number.

6.2. Effect of New Versions.
Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Netscape. No one other than Netscape has the right to modify the terms applicable to Covered Code created under this License.

6.3. Derivative Works.
If You create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), You must (a) rename Your license so that the phrases "Mozilla", "MOZILLAPL", "MOZPL", "Netscape", "MPL", "NPL" or any confusingly similar phrase do not appear in your license (except to note that your license differs from this License) and (b) otherwise make it clear that Your version of the license contains terms which differ from the Mozilla Public License and Netscape Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. TERMINATION.

8.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

8.2. If You initiate litigation by asserting a patent infringement

claim (excluding declatory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You file such action is referred to as "Participant") alleging that:

(a) such Participant's Contributor Version directly or indirectly infringes any patent, then any and all rights granted by such Participant to You under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively, unless if within 60 days after receipt of notice You either: (i) agree in writing to pay Participant a mutually agreeable reasonable royalty for Your past and future use of Modifications made by such Participant, or (ii) withdraw Your litigation claim with respect to the Contributor Version against such Participant. If within 60 days of notice, a reasonable royalty and payment arrangement are not mutually agreed upon in writing by the parties or the litigation claim is not withdrawn, the rights granted by Participant to You under Sections 2.1 and/or 2.2 automatically terminate at the expiration of the 60 day notice period specified above.

(b) any software, hardware, or device, other than such Participant's Contributor Version, directly or indirectly infringes any patent, then any rights granted to You by such Participant under Sections 2.1(b) and 2.2(b) are revoked effective as of the date You first made, used, sold, distributed, or had made, Modifications made by that Participant.

8.3. If You assert a patent infringement claim against Participant alleging that such Participant's Contributor Version directly or indirectly infringes any patent where such claim is resolved (such as by license or settlement) prior to the initiation of patent infringement litigation, then the reasonable value of the licenses granted by such Participant under Sections 2.1 or 2.2 shall be taken into account in determining the amount or value of any payment or license.

8.4. In the event of termination under Sections 8.1 or 8.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or any distributor hereunder prior to termination shall survive termination.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL,

WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein.

11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in the United States of America, any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

13. MULTIPLE-LICENSED CODE.

Initial Developer may designate portions of the Covered Code as "Multiple-Licensed". "Multiple-Licensed" means that the Initial Developer permits you to utilize portions of the Covered Code under Your choice of the NPL or the alternative licenses, if any, specified by the Initial Developer in the file described in Exhibit A.

EXHIBIT A -Mozilla Public License.

"The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is _____.

The Initial Developer of the Original Code is _____. Portions created by _____ are Copyright (C) _____ _____. All Rights Reserved.

Contributor(s): _____.

Alternatively, the contents of this file may be used under the terms of the _____ license (the "[___] License"), in which case the provisions of [_____] License are applicable instead of those above. If you wish to allow use of your version of this file only under the terms of the [____] License and not to allow others to use your version of this file under the MPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the [___] License. If you do not delete the provisions above, a recipient may use your version of this file under either the MPL or the [___] License."

[NOTE: The text of this Exhibit A may differ slightly from the text of the notices in the Source Code files of the Original Code. You should use the text of this Exhibit A rather than the text found in the Original Code Source Code for Your Modifications.]

```
/* -*- Mode: java; tab-width: 8; indent-tabs-mode: nil; c-basic-offset: 4 -*-
*
* ***** BEGIN LICENSE BLOCK *****
* Version: MPL 1.1/GPL 2.0
```

*
* The contents of this file are subject to the Mozilla Public License Version
* 1.1 (the "License"); you may not use this file except in compliance with
* the License. You may obtain a copy of the License at
* http://www.mozilla.org/MPL/
*
* Software distributed under the License is distributed on an "AS IS" basis,
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License
* for the specific language governing rights and limitations under the
* License.
*
* The Original Code is Rhino code, released
* May 6, 1999.
*
* The Initial Developer of the Original Code is
* Netscape Communications Corporation.
* Portions created by the Initial Developer are Copyright (C) 1997-1999
* the Initial Developer. All Rights Reserved.
*
* Contributor(s):
*   Roger Lawrence
*
* Alternatively, the contents of this file may be used under the terms of
* the GNU General Public License Version 2 or later (the "GPL"), in which
* case the provisions of the GPL are applicable instead of those above. If
* you wish to allow use of your version of this file only under the terms of
* the GPL and not to allow others to use your version of this file under the
* MPL, indicate your decision by deleting the provisions above and replacing
* them with the notice and other provisions required by the GPL. If you do
* not delete the provisions above, a recipient may use your version of this
* file under either the MPL or the GPL.
*
* ***** END LICENSE BLOCK *****

1.43 Commons Lang 2.6

1.43.1 Available under license :

Apache License, Version 2.0
FoundationProjectsPeopleGet InvolvedDownloadSupport ApacheHome » Licenses
Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to

You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.
5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work
To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

:::text
Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache Commons Lang
Copyright 2001-2011 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.44 Commons Logging 1.1.1

1.44.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain

separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the

origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache Commons Logging
Copyright 2003-2007 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).
<!--

Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements. See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

-->

1.45 Commons Pool 1.6

1.45.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to

communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of

the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and
limitations under the License.
Apache Commons Pool
Copyright 2001-2012 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.46 commons-beanutils 1.9.3

1.46.1 Available under license :

Apache Commons BeanUtils
Copyright 2000-2016 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical

transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable

by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use,

reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.47 commons-beanutils-core 1.7.0

1.47.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all

other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and

subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed

as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the

Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.48 commons-codec 1.6

1.48.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes

of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You

meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,

except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache Commons Codec
Copyright 2002-2011 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

---
src/test/org/apache/commons/codec/language/DoubleMetaphoneTest.java contains
test data from http://aspell.sourceforge.net/test/batch0.tab.

Copyright (C) 2002 Kevin Atkinson (kevina@gnu.org). Verbatim copying
and distribution of this entire article is permitted in any medium,
provided this notice is preserved.

1.49 commons-collections 3.2.1

1.49.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all

other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and

subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed

as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the

Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache Commons Collections

Copyright 2001-2008 The Apache Software Foundation

This product includes software developed by

The Apache Software Foundation (http://www.apache.org/).

1.50 commons-configuration 1.2

1.51 commons-configuration 1.4

1.51.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a

cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,

any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache Jakarta Commons Configuration
Copyright 2001-2007 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.52 commons-dbcp 1.4

1.52.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all

other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and

subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed

as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the

Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache Commons DBCP
Copyright 2001-2010 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.53 commons-digester 1.7

1.53.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes

of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You

meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,

except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.54 commons-discovery 0.2

1.54.1 Notifications :

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

1.54.2 Available under license :

```
/*
* $Header: /home/cvs/jakarta-commons/discovery/LICENSE.txt,v 1.1 2002/07/25 02:36:45 jvanzyl Exp $
* $Revision: 1.1 $
* $Date: 2002/07/25 02:36:45 $
*
* ====================================================================
*
* The Apache Software License, Version 1.1
*
* Copyright (c) 1999-2001 The Apache Software Foundation.  All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
```

```
* 3. The end-user documentation included with the redistribution, if
*    any, must include the following acknowlegement:
*       "This product includes software developed by the
*        Apache Software Foundation (http://www.apache.org/)."
*    Alternately, this acknowlegement may appear in the software itself,
*    if and wherever such third-party acknowlegements normally appear.
*
* 4. The names "The Jakarta Project", "Commons", and "Apache Software
*    Foundation" must not be used to endorse or promote products derived
*    from this software without prior written permission. For written
*    permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache"
*    nor may "Apache" appear in their names without prior written
*    permission of the Apache Group.
*
* THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* ====================================================================
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation.  For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*
*/
```

1.55 commons-httpclient 3.1

1.55.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to

communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of

the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Apache Jakarta HttpClient
Copyright 1999-2007 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).
Apache Jakarta HttpClient
Copyright 1999-2007 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not

pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special,

incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

1.56 commons-lang3 3.3.1

1.56.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity

exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided

that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,

or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache Commons Lang
Copyright 2001-2011 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

This product includes software from the Spring Framework,
under the Apache License 2.0 (see: StringUtils.containsWhitespace())

1.57 commons-lang3 3.5

1.57.1 Available under license :

Apache Commons Lang
Copyright 2001-2016 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

This product includes software from the Spring Framework,
under the Apache License 2.0 (see: StringUtils.containsWhitespace())

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work

(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses

granted to You under this License for that Work shall terminate
as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]"

replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.58 commons-logging 1.0.3

1.58.1 Notifications :

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

1.58.2 Available under license :

```
/*
* $Header: /home/cvspublic/jakarta-commons/logging/LICENSE.txt,v 1.2 2003/04/06 20:37:31 rdonkin Exp $
* $Revision: 1.2 $
* $Date: 2003/04/06 20:37:31 $
*
* ========================================================================
*
* The Apache Software License, Version 1.1
*
* Copyright (c) 1999-2003 The Apache Software Foundation.  All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
```

```
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. The end-user documentation included with the redistribution, if
*    any, must include the following acknowlegement:
*       "This product includes software developed by the
*        Apache Software Foundation (http://www.apache.org/)."
*    Alternately, this acknowlegement may appear in the software itself,
*    if and wherever such third-party acknowlegements normally appear.
*
* 4. The names "The Jakarta Project", "Commons", and "Apache Software
*    Foundation" must not be used to endorse or promote products derived
*    from this software without prior written permission. For written
*    permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache"
*    nor may "Apache" appear in their names without prior written
*    permission of the Apache Group.
*
* THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* ====================================================================
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation.  For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*
*/
```

1.59 cxf 2.5.2

1.59.1 Available under license :

(c) 2001-2006 BEA Systems Inc., International Business Machines Corporation, Microsoft Corporation, Inc., SAP AG, Sonic Software, and VeriSign, Inc. All rights reserved.

Permission to copy and display the WS-Policy Specification (the "Specification", which includes WSDL and schema documents), in any medium without fee or royalty is hereby granted, provided that you include the following on ALL copies of the WS-Policy Specification, that you make:

1. A link or URL to the WS-Policy Specification at one of the Authors' websites
2. The copyright notice as shown in the WS-Policy Specification.

BEA Systems, IBM, Microsoft, SAP, Sonic Software, and VeriSign (collectively, the "Authors") each agree to grant you a license, under royalty-free and otherwise reasonable, non-discriminatory terms and conditions, to their respective essential patent claims that they deem necessary to implement the WS-Policy Specification.

THE WS-POLICY SPECIFICATION IS PROVIDED "AS IS," AND THE AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE WS-POLICY SPECIFICATION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE WS-POLICY SPECIFICATION.

The name and trademarks of the Authors may NOT be used in any manner, including advertising or publicity pertaining to the WS-Policy Specification or its contents without specific, written prior permission. Title to copyright in the WS-Policy Specification will at all times remain with the Authors.

No other rights are granted by implication, estoppel or otherwise.
The BSD License

Copyright (c) <YEAR>, <OWNER>
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
* Neither the name of the <ORGANIZATION> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND

ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Copyright (c) 2000-2005 INRIA, France Telecom
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holders nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Common Public License Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any

Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its

Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND

NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. Contributor means each individual or entity that creates or contributes to the creation of Modifications.

1.2. Contributor Version means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. Covered Software means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. Executable means the Covered Software in any form other than Source Code.

1.5. Initial Developer means the individual or entity that first makes Original Software available under this License.

1.6. Larger Work means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. License means this document.

1.8. Licensable means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. Original Software means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. Patent Claims means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. Source Code means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. You (or Your) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, You includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, control means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making,

using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications

available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipients rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and

may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN AS IS BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as Participant) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL

INJURY RESULTING FROM SUCH PARTYS NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a commercial item, as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of commercial computer software (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and commercial computer software documentation as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdictions conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or

indirectly, out of its utilization of rights under this License
and You agree to work with Initial Developer and Contributors to
distribute such responsibility on an equitable basis. Nothing
herein is intended or shall be deemed to constitute any admission
of liability.
/*
$Id: LICENSE.txt,v 1.5 2006/02/05 21:49:04 elharo Exp $

Copyright 2003-2006 The Werken Company. All Rights Reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

* Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

* Neither the name of the Jaxen Project nor the names of its
contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*/
/*--

$Id: LICENSE.txt,v 1.11 2004/02/06 09:32:57 jhunter Exp $

Copyright (C) 2000-2004 Jason Hunter & Brett McLaughlin.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions, and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions, and the disclaimer that follows
   these conditions in the documentation and/or other materials
   provided with the distribution.

3. The name "JDOM" must not be used to endorse or promote products
   derived from this software without prior written permission. For
   written permission, please contact <request_AT_jdom_DOT_org>.

4. Products derived from this software may not be called "JDOM", nor
   may "JDOM" appear in their name, without prior written permission
   from the JDOM Project Management <request_AT_jdom_DOT_org>.

In addition, we request (but do not require) that you include in the
end-user documentation provided with the redistribution and/or in the
software itself an acknowledgement equivalent to the following:
    "This product includes software developed by the
     JDOM Project (http://www.jdom.org/)."
Alternatively, the acknowledgment may be graphical using the logos
available at http://www.jdom.org/images/logos.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

This software consists of voluntary contributions made by many
individuals on behalf of the JDOM Project and was originally
created by Jason Hunter <jhunter_AT_jdom_DOT_org> and
Brett McLaughlin <brett_AT_jdom_DOT_org>. For more information
on the JDOM Project, please see <http://www.jdom.org/>.

*/

License for WSDL Schema Files

The Authors grant permission to copy and distribute the WSDL Schema Files in any medium without fee or royalty as long as this notice and license are distributed with them. The originals of these files can be located at:

http://schemas.xmlsoap.org/wsdl/2003-02-11.xsd

THESE SCHEMA FILES ARE PROVIDED "AS IS," AND THE AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, REGARDING THESE FILES, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT OR TITLE. THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THESE FILES.

The name and trademarks of the Authors may NOT be used in any manner, including advertising or publicity pertaining to these files or any program or service that uses these files, written prior permission. Title to copyright in these files will at all times remain with the Authors.

No other rights are granted by implication, estoppel or otherwise.
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta name="generator" content="HTML Tidy, see www.w3.org" />
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1" />
<link rel="stylesheet" type="text/css"
href="/StyleSheets/base.css" />
<title>W3C IPR SOFTWARE NOTICE</title>
</head>
<body text="#000000" bgcolor="#FFFFFF">
<h1>W3C® SOFTWARE NOTICE AND LICENSE</h1>

<h3>Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/</h3>

<p>This W3C work (including software, documents, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will

comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software
and its documentation, with or without modification, for any
purpose and without fee or royalty is hereby granted, provided that
you include the following on ALL copies of the software and
documentation or portions thereof, including modifications, that
you make:

The full text of this NOTICE in a location viewable to users of
the redistributed or derivative work.

Any pre-existing intellectual property disclaimers, notices, or
terms and conditions. If none exist, a short notice of the
following form (hypertext is preferred, text is permitted) should
be used within the body of any redistributed or derivative code:
"Copyright © [$date-of-software] <a
href="http://www.w3.org/">World Wide Web Consortium, (<a
href="http://www.lcs.mit.edu/">Massachusetts Institute of
Technology, Institut National de
Recherche en Informatique et en Automatique, <a
href="http://www.keio.ac.jp/">Keio University). All Rights
Reserved. http://www.w3.org/Consortium/Legal/"

Notice of any changes or modifications to the W3C files,
including the date changes were made. (We recommend you provide
URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND
COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE
USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD
PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT,
SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE
SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in
advertising or publicity pertaining to the software without
specific, written prior permission. Title to copyright in this
software and any associated documentation will at all times remain
with copyright holders.

_____

This formulation of W3C's notice and license became active on
August 14 1998 so as to improve compatibility with GPL. This
version ensures that W3C software licensing terms are no more
restrictive than GPL and consequently W3C software may be
distributed in GPL packages. See the <a
href="copyright-software-19980519.html">older formulation for
the policy prior to this date. Please see our <a
href="IPR-FAQ.html">Copyright FAQ for common questions about
using materials from our site, including specific terms and
conditions for packages like libwww, Amaya, and Jigsaw. Other
questions about this notice can be directed to <a
href="mailto:site-policy@w3.org">site-policy@w3.org.

<address>webmaster
(last updated $Date: 2006-11-08 08:52:14 -0500 (Wed, 08 Nov 2006) $)</address>
</body>
</html>
This Product also includes software developed by David Heinemeier Hansson.
(http://dev.rubyonrails.org/browser/trunk/activesupport/lib/active_support/inflections.rb)

This product includes software Copyright University of Southampton IT Innovation Centre, 2009
(http://www.it-innovation.soton.ac.uk).
```
/**
* Licensed to the Apache Software Foundation (ASF) under one
* or more contributor license agreements. See the NOTICE file
* distributed with this work for additional information
* regarding copyright ownership. The ASF licenses this file
* to you under the Apache License, Version 2.0 (the
* "License"); you may not use this file except in compliance
* with the License. You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
* software distributed under the License is distributed on an
* "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
* KIND, either express or implied. See the License for the
* specific language governing permissions and limitations
* under the License.
*/
package org.apache.cxf.rs.security.oauth.data;

/**
* Base permission description which is visible to
```

```
* authorization handlers
* @see OAuthAuthorizationData
*/
public class Permission {
  private String permission;
  private String description;

  public Permission() {

  }

  public Permission(String permission, String description) {
     this.description = description;
     this.permission = permission;
  }

  public String getDescription() {
     return description;
  }

  public void setDescription(String description) {
     this.description = description;
  }

  public String getPermission() {
     return permission;
  }

  public void setPermission(String permission) {
     this.permission = permission;
  }
}
```

Portions of the file cxf-utils.js derives from code marked:
This code was written by Tyler Akins and has been placed in the
public domain. It would be nice if you left this header intact.
Base64 code from Tyler Akins -- http://rumkin.com

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions, and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright

notice, this list of conditions, and the disclaimer that follows
these conditions in the documentation and/or other materials
provided with the distribution.

3. The name "JDOM" must not be used to endorse or promote products
derived from this software without prior written permission. For
written permission, please contact <request_AT_jdom_DOT_org>.

4. Products derived from this software may not be called "JDOM", nor
may "JDOM" appear in their name, without prior written permission
from the JDOM Project Management <request_AT_jdom_DOT_org>.

In addition, we request (but do not require) that you include in the
end-user documentation provided with the redistribution and/or in the
software itself an acknowledgement equivalent to the following:
"This product includes software developed by the
JDOM Project (http://www.jdom.org/)."
Alternatively, the acknowledgment may be graphical using the logos
available at http://www.jdom.org/images/logos.

THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

This software consists of voluntary contributions made by many
individuals on behalf of the JDOM Project and was originally
created by Jason Hunter <jhunter_AT_jdom_DOT_org> and
Brett McLaughlin <brett_AT_jdom_DOT_org>. For more information
on the JDOM Project, please see <http://www.jdom.org/>.

The product contains code (StaxBuilder.java) that is
Copyright (C) 2000-2004 Jason Hunter & Brett McLaughlin.
All rights reserved.
See the NOTICE.jdom file for additional information
Java classes (source and binary) under org.apache.cxf.jaxws.javaee
are generated from schema available here:
(http://java.sun.com/xml/ns/javaee/javaee_5.xsd)
File json2.min.js has been placed in the public domain.

Portions of the file uuid.min.js derives from code marked:
/*
Math.uuid.js (v1.4)
http://www.broofa.com
mailto:robert@broofa.com

Copyright (c) 2010 Robert Kieffer
Dual licensed under the MIT and GPL licenses.
*/
##
Licensed to the Apache Software Foundation (ASF) under one
or more contributor license agreements. See the NOTICE file
distributed with this work for additional information
regarding copyright ownership. The ASF licenses this file
to you under the Apache License, Version 2.0 (the
"License"); you may not use this file except in compliance
with the License. You may obtain a copy of the License at
##
http://www.apache.org/licenses/LICENSE-2.0
##
Unless required by applicable law or agreed to in writing,
software distributed under the License is distributed on an
"AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, either express or implied. See the License for the
specific language governing permissions and limitations
under the License.
##
$Date: 2008-03-09 23:17:06 -0700 (Sun, 09 Mar 2008) $ $Rev: 635446 $
##

Apache CXF includes a number of components and libraries with separate
copyright notices and license terms. Your use of those components are
subject to the terms and conditions of the following licenses.

#set ($apacheTxt = "The Apache Software License, Version 2.0")

#foreach ($project in $projects)
#foreach ($license in $project.licenses)
#if (! ($apacheTxt == $license.name))

$project.name #if ($project.url)($project.url)#end $project.artifact
  License: $license.name #if ($license.url) ($license.url)#end

#end
#end
#end

This product also includes schemas and specification developed by:
- the W3C consortium (http://www.w3c.org)
(http://www.w3.org/XML/1998/namespace)

This product also includes WS-* schemas developed by International
Business Machines Corporation, Microsoft Corporation, BEA Systems,
TIBCO Software, SAP AG, Sonic Software, and VeriSign
(http://schemas.xmlsoap.org/wsdl/2003-02-11.xsd)
(http://schemas.xmlsoap.org/ws/2004/08/addressing/)
(http://schemas.xmlsoap.org/wsdl/http)
(http://schemas.xmlsoap.org/ws/2005/02/rm/wsrm.xsd)
(http://www.w3.org/2005/08/addressing/ws-addr.xsd)
(http://www.w3.org/TR/ws-metadata-exchange/)
(http://schemas.xmlsoap.org/ws/2004/09/mex/)
(http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html)

The product contains code (StaxBuilder.java) that is
Copyright (C) 2000-2004 Jason Hunter & Brett McLaughlin.
All rights reserved.
See the NOTICE.jdom file for additional information

Java classes (source and binary) under org.apache.cxf.jaxws.javaee
are generated from schema available here:
(http://java.sun.com/xml/ns/javaee/javaee_5.xsd)

This Product also includes software developed by David Heinemeier Hansson.
(http://dev.rubyonrails.org/browser/trunk/activesupport/lib/active_support/inflections.rb)

This product includes software Copyright University of Southampton IT Innovation Centre, 2009
(http://www.it-innovation.soton.ac.uk).

This product also includes MTOSI wsdl and schemas developed by the
TeleManagement Forum (http://www.tmforum.org/browse.aspx). The original
MTOSI wsdl and schemas can be download from
(http://sourceforge.net/projects/mtosi-ri)

Portions of the included XmlSchema library are Copyright 2006 International Business Machines Corp.

Portions of the included xml-apis library were originally based on the following:
- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
- software copyright (c) 2000 World Wide Web Consortium, http://www.w3.org

Portions of the included xmlbeans library were originally based on the following:
- software copyright (c) 2000-2003, BEA Systems, <http://www.bea.com/>.

Portions of the file cxf-utils.js derives from code marked:
This code was written by Tyler Akins and has been placed in the

public domain. It would be nice if you left this header intact.
Base64 code from Tyler Akins -- http://rumkin.com

Additional copyright notices and license terms applicable are
present in the licenses directory of this distribution.

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the

editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the

Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the

same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache CXF includes a number of components and libraries with separate
copyright notices and license terms. Your use of those components are
subject to the terms and conditions of the following licenses.

AntLR Parser Generator (http://www.antlr.org/) antlr:antlr:jar:2.7.7
License: BSD License (http://www.antlr.org/license.html)

AOP alliance (http://aopalliance.sourceforge.net) aopalliance:aopalliance:jar:1.0
License: Public Domain

ASM (http://asm.objectweb.org/asm/asm) asm:asm:jar:3.3:compile
License: BSD (http://asm.ow2.org/license.html)

ICU4J (http://www.icu-project.org/) com.ibm.icu:icu4j:jar:4.0.1
License: ICU License (http://source.icu-project.org/repos/icu/icu/trunk/license.html)

MSV XML Schema Datatype Library (http://nexus.sonatype.org/oss-repository-hosting.html/xsdlib)
com.sun.msv.datatype.xsd:xsdlib:bundle:2010.1
License: BSD (http://www.opensource.org/licenses/bsd-license.php)

Sun JAXB Reference Implementation Runtime (http://jaxb.java.net/jaxb-impl) com.sun.xml.bind:jaxb-
impl:jar:2.2.4-1:compile
License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0
(http://www.sun.com/cddl/cddl.html)

Sun JAXB Reference Implementation Tools (http://jaxb.java.net/jaxb-xjc) com.sun.xml.bind:jaxb-xjc:jar:2.2.4-
1:compile
License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0
(http://www.sun.com/cddl/cddl.html)

Sun SAAJ Reference Implementation (http://java.net/saaj-impl/saaj-impl) com.sun.xml.messaging.saaj:saaj-impl:jar:1.3.12:compile
License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0 (http://www.sun.com/cddl/cddl.html)

ISO Relax (http://iso-relax.sourceforge.net/) isorelax:isorelax:jar:20030108:compile
License: MIT (http://www.opensource.org/licenses/mit-license.html)

JSR 311 API (https://jsr311.dev.java.net/) javax.ws.rs:jsr311-api:jar:1.1.1:compile
License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0 (http://www.sun.com/cddl/cddl.html)

Java Architecture for XML Binding (JAXB API) (http://jaxb.java.net/jaxb-api) javax.xml.bind:jaxb-api:jar:2.2.3:compile
License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0 (http://www.sun.com/cddl/cddl.html)

Sun SAAJ API (http://java.net/saaj-api/saaj-api) javax.xml.soap:saaj-api:jar:1.3.4:compile
License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0 (http://www.sun.com/cddl/cddl.html)

Joda time (http://joda-time.sourceforge.net) joda-time:joda-time:jar:1.6.2
License: Apache 2 (http://www.apache.org/licenses/LICENSE-2.0.txt)

MSV Core (http://msv.java.net/msv-core) net.java.dev.msv:msv-core:bundle:2011.1
License: BSD ()

Stax2 API (http://woodstox.codehaus.org/StAX2) org.codehaus.woodstox:stax2-api:jar:3.1.1
License: The BSD License (http://www.opensource.org/licenses/bsd-license.php)

Jetty :: Continuation (http://www.eclipse.org/jetty/jetty-continuation) org.eclipse.jetty:jetty-continuation:jar:7.5.4.v20111024
License: Apache Software License - Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

Jetty :: Continuation (http://www.eclipse.org/jetty/jetty-continuation) org.eclipse.jetty:jetty-continuation:jar:7.5.4.v20111024
License: Eclipse Public License - Version 1.0 (http://www.eclipse.org/org/documents/epl-v10.php)

Jetty :: Http Utility (http://www.eclipse.org/jetty/jetty-http) org.eclipse.jetty:jetty-http:jar:7.5.4.v20111024
License: Apache Software License - Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

Jetty :: Http Utility (http://www.eclipse.org/jetty/jetty-http) org.eclipse.jetty:jetty-http:jar:7.5.4.v20111024
License: Eclipse Public License - Version 1.0 (http://www.eclipse.org/org/documents/epl-v10.php)

Jetty :: IO Utility (http://www.eclipse.org/jetty/jetty-io) org.eclipse.jetty:jetty-io:jar:7.5.4.v20111024
License: Apache Software License - Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

Jetty :: IO Utility (http://www.eclipse.org/jetty/jetty-io) org.eclipse.jetty:jetty-io:jar:7.5.4.v20111024
License: Eclipse Public License - Version 1.0 (http://www.eclipse.org/org/documents/epl-v10.php)

Jetty :: Security (http://www.eclipse.org/jetty/jetty-security) org.eclipse.jetty:jetty-security:jar:7.5.4.v20111024
License: Apache Software License - Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

Jetty :: Security (http://www.eclipse.org/jetty/jetty-security) org.eclipse.jetty:jetty-security:jar:7.5.4.v20111024
License: Eclipse Public License - Version 1.0 (http://www.eclipse.org/org/documents/epl-v10.php)

Jetty :: Server Core (http://www.eclipse.org/jetty) org.eclipse.jetty:jetty-server:jar:7.5.4.v20111024
License: Apache Software License - Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

Jetty :: Server Core (http://www.eclipse.org/jetty) org.eclipse.jetty:jetty-server:jar:7.5.4.v20111024
License: Eclipse Public License - Version 1.0 (http://www.eclipse.org/org/documents/epl-v10.php)

Jetty :: Utilities (http://www.eclipse.org/jetty/jetty-util) org.eclipse.jetty:jetty-util:jar:7.5.4.v20111024
License: Apache Software License - Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

Jetty :: Utilities (http://www.eclipse.org/jetty/jetty-util) org.eclipse.jetty:jetty-util:jar:7.5.4.v20111024
License: Eclipse Public License - Version 1.0 (http://www.eclipse.org/org/documents/epl-v10.php)

MIME streaming extension org.jvnet:mimepull:jar:1.4
License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0
(http://www.opensource.org/licenses/cddl1.php)

MIME streaming extension org.jvnet:mimepull:jar:1.4
License: GPLv2 with classpath exception (http://www.gnu.org/software/classpath/license.html)

OpenSAML-J (http://opensaml.org/) org.opensaml:opensaml:jar:2.5.1-1
License: Apache 2 (http://www.apache.org/licenses/LICENSE-2.0.txt)

OpenWS (http://opensaml.org/) org.opensaml:openws:jar:1.4.2-1
License: Apache 2 (http://www.apache.org/licenses/LICENSE-2.0.txt)

XMLTooling-J (http://opensaml.org/) org.opensaml:xmltooling:jar:1.3.2-1
License: Apache 2 (http://www.apache.org/licenses/LICENSE-2.0.txt)

Simple Logging Facade for Java - API (http://www.slf4j.org/slf4j-api) org.slf4j:slf4j-api:jar:1.6.2:compile
License: MIT License (http://www.slf4j.org/license.html)

Simple Logging Facade for Java - JDK Logging (http://www.slf4j.org/slf4j-jdk14) org.slf4j:slf4j-
jdk14:jar:1.6.2:compile
License: MIT License (http://www.slf4j.org/license.html)

RELAX NG Datatype (http://sourceforge.net/projects/relaxng/)
relaxngDatatype:relaxngDatatype:jar:20020414:compile
License: BSD (http://www.opensource.org/licenses/bsd-license.php)

Rhino (http://www.mozilla.org/rhino/) rhino:js:jar:1.7R2
License: Mozilla Public License (http://www.mozilla.org/MPL/MPL-1.1.html)

WSDL4J (http://sf.net/projects/wsdl4j) wsdl4j:wsdl4j:jar:1.6.2
License: CPL (http://www.opensource.org/licenses/cpl1.0.txt)

Apache CXF
Copyright 2006-2012 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

This product also includes schemas and specification developed by:
- the W3C consortium (http://www.w3c.org)
(http://www.w3.org/XML/1998/namespace)

This product also includes WS-* schemas developed by International
Business Machines Corporation, Microsoft Corporation, BEA Systems,
TIBCO Software, SAP AG, Sonic Software, and VeriSign
(http://schemas.xmlsoap.org/wsdl/2003-02-11.xsd)
(http://schemas.xmlsoap.org/ws/2004/08/addressing/)
(http://schemas.xmlsoap.org/wsdl/http)
(http://schemas.xmlsoap.org/ws/2005/02/rm/wsrm.xsd)
(http://www.w3.org/2005/08/addressing/ws-addr.xsd)
(http://www.w3.org/TR/ws-metadata-exchange/)
(http://schemas.xmlsoap.org/ws/2004/09/mex/)
(http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html)

The product contains code (StaxBuilder.java) that is
Copyright (C) 2000-2004 Jason Hunter & Brett McLaughlin.
All rights reserved.
See the NOTICE.jdom file for additional information

Java classes (source and binary) under org.apache.cxf.jaxws.javaee
are generated from schema available here:
(http://java.sun.com/xml/ns/javaee/javaee_5.xsd)

This Product also includes software developed by David Heinemeier Hansson.
(http://dev.rubyonrails.org/browser/trunk/activesupport/lib/active_support/inflections.rb)

This product includes software Copyright University of Southampton IT Innovation Centre, 2009
(http://www.it-innovation.soton.ac.uk).

This product also includes MTOSI wsdl and schemas developed by the
TeleManagement Forum (http://www.tmforum.org/browse.aspx). The original
MTOSI wsdl and schemas can be download from
(http://sourceforge.net/projects/mtosi-ri)

Portions of the included XmlSchema library are Copyright 2006 International Business Machines Corp.

Portions of the included xml-apis library were originally based on the following:
- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
- software copyright (c) 2000 World Wide Web Consortium, http://www.w3.org

Portions of the included xmlbeans library were originally based on the following:
- software copyright (c) 2000-2003, BEA Systems, <http://www.bea.com/>.

Portions of the file cxf-utils.js derives from code marked:
This code was written by Tyler Akins and has been placed in the
public domain. It would be nice if you left this header intact.
Base64 code from Tyler Akins -- http://rumkin.com

Additional copyright notices and license terms applicable are
present in the licenses directory of this distribution.
Copyright 2002-2004 BEA Systems Inc., International Business Machines Corporation,
Microsoft Corporation, Inc, SAP AG, and Sun Microsystems, Inc.. All rights reserved.

Permission to copy, display, perform, modify and distribute the WS-Addressing Specification,
and to authorize others to do the foregoing, in any medium without fee or royalty is hereby
granted for the purpose of developing and evaluating the WS-Addressing Specification.

BEA, IBM, Microsoft, SAP AG, and Sun Microsystems (collectively, the "Authors") each agree
to grant a license to third parties, under royalty-free and otherwise reasonable,
non-discriminatory terms and conditions, to their respective essential patent claims that
they deem necessary to implement the WS-Addressing Specification.

DISCLAIMERS:

THE WS-Addressing Specification IS PROVIDED "AS IS", AND THE AUTHORS MAKE NO
REPRESENTATIONS
OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE;
THAT THE
CONTENTS OF THE WS-Addressing Specification IS SUITABLE FOR ANY PURPOSE; NOR THAT THE
IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS,
COPYRIGHTS,
TRADEMARKS OR OTHER RIGHTS.

THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR
CONSEQUENTIAL
DAMAGES ARISING OUT OF ANY USE OF THE WS-Addressing Specification OR THE PERFORMANCE OR

IMPLEMENTATION OF THE CONTENTS THEREOF.

You may remove these disclaimers from your modified versions of the WS-Addressing Specification provided that you effectively disclaim all warranties and liabilities on behalf of all copyright holders in the copies of any such modified versions you distribute.

The name and trademarks of the Authors may NOT be used in any manner, including advertising or publicity pertaining to the WS-Addressing Specification or its contents without specific, written prior permission. Title to copyright in the WS-Addressing Specification will at all times remain with the Authors.

No other rights are granted by implication, estoppel or otherwise.
OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.
OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.
Copyright OASIS Open 2002-2004. All Rights Reserved.
This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself does not be modified in any way, such as by removing the copyright notice or references to OASIS, except as needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it into languages other than English.
The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.
This document and the information contained herein is provided on an AS IS basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Eclipse Public License - Version 1.0</title>
<style type="text/css">
 body {
   size: 8.5in 11.0in;
   margin: 0.25in 0.5in 0.25in 0.5in;
   tab-interval: 0.5in;

```
  }
 p {
  margin-left: auto;
  margin-top:  0.5em;
  margin-bottom: 0.5em;
  }
 p.list {
  margin-left: 0.5in;
  margin-top:  0.05em;
  margin-bottom: 0.05em;
  }
 </style>

</head>

<body lang="EN-US">

<h2>Eclipse Public License - v 1.0</h2>

<p>THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE
PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR
DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS
AGREEMENT.</p>

<p><b>1. DEFINITIONS</b></p>

<p>"Contribution" means:</p>

<p class="list">a) in the case of the initial Contributor, the initial
code and documentation distributed under this Agreement, and</p>
<p class="list">b) in the case of each subsequent Contributor:</p>
<p class="list">i) changes to the Program, and</p>
<p class="list">ii) additions to the Program;</p>
<p class="list">where such changes and/or additions to the Program
originate from and are distributed by that particular Contributor. A
Contribution 'originates' from a Contributor if it was added to the
Program by such Contributor itself or anyone acting on such
Contributor's behalf. Contributions do not include additions to the
Program which: (i) are separate modules of software distributed in
conjunction with the Program under their own license agreement, and (ii)
are not derivative works of the Program.</p>

<p>"Contributor" means any person or entity that distributes
the Program.</p>

<p>"Licensed Patents" mean patent claims licensable by a
Contributor which are necessarily infringed by the use or sale of its
Contribution alone or when combined with the Program.</p>
```

"Program" means the Contributions distributed in accordance
with this Agreement.

"Recipient" means anyone who receives the Program under
this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each
Contributor hereby grants Recipient a non-exclusive, worldwide,
royalty-free copyright license to reproduce, prepare derivative works
of, publicly display, publicly perform, distribute and sublicense the
Contribution of such Contributor, if any, and such derivative works, in
source code and object code form.

b) Subject to the terms of this Agreement, each
Contributor hereby grants Recipient a non-exclusive, worldwide,
royalty-free patent license under Licensed Patents to make, use, sell,
offer to sell, import and otherwise transfer the Contribution of such
Contributor, if any, in source code and object code form. This patent
license shall apply to the combination of the Contribution and the
Program if, at the time the Contribution is added by the Contributor,
such addition of the Contribution causes such combination to be covered
by the Licensed Patents. The patent license shall not apply to any other
combinations which include the Contribution. No hardware per se is
licensed hereunder.

c) Recipient understands that although each Contributor
grants the licenses to its Contributions set forth herein, no assurances
are provided by any Contributor that the Program does not infringe the
patent or other intellectual property rights of any other entity. Each
Contributor disclaims any liability to Recipient for claims brought by
any other entity based on infringement of intellectual property rights
or otherwise. As a condition to exercising the rights and licenses
granted hereunder, each Recipient hereby assumes sole responsibility to
secure any other intellectual property rights needed, if any. For
example, if a third party patent license is required to allow Recipient
to distribute the Program, it is Recipient's responsibility to acquire
that license before distributing the Program.

d) Each Contributor represents that to its knowledge it
has sufficient copyright rights in its Contribution, if any, to grant
the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code

form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor

("Commercial Contributor") hereby agrees to defend and
indemnify every other Contributor ("Indemnified Contributor")
against any losses, damages and costs (collectively "Losses")
arising from claims, lawsuits and other legal actions brought by a third
party against the Indemnified Contributor to the extent caused by the
acts or omissions of such Commercial Contributor in connection with its
distribution of the Program in a commercial product offering. The
obligations in this section do not apply to any claims or Losses
relating to any actual or alleged intellectual property infringement. In
order to qualify, an Indemnified Contributor must: a) promptly notify
the Commercial Contributor in writing of such claim, and b) allow the
Commercial Contributor to control, and cooperate with the Commercial
Contributor in, the defense and any related settlement negotiations. The
Indemnified Contributor may participate in any such claim at its own
expense.

For example, a Contributor might include the Program in a commercial
product offering, Product X. That Contributor is then a Commercial
Contributor. If that Commercial Contributor then makes performance
claims, or offers warranties related to Product X, those performance
claims and warranties are such Commercial Contributor's responsibility
alone. Under this section, the Commercial Contributor would have to
defend claims against the other Contributors related to those
performance claims and warranties, and if a court requires any other
Contributor to pay any damages as a result, the Commercial Contributor
must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS
PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION,
ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY
OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely
responsible for determining the appropriateness of using and
distributing the Program and assumes all risks associated with its
exercise of rights under this Agreement , including but not limited to
the risks and costs of program errors, compliance with applicable laws,
damage to or loss of data, programs or equipment, and unavailability or
interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT
NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING
WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING

NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

</body>

</html>

MOZILLA PUBLIC LICENSE
Version 1.1

---

1. Definitions.

1.0.1. "Commercial Use" means distribution or otherwise making the Covered Code available to a third party.

1.1. "Contributor" means each entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. "Covered Code" means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof.

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. "License" means this document.

1.8.1. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or

subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. "Original Code" means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.10.1. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.11. "Source Code" means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated interface definition files, scripts used to control compilation and installation of an Executable, or source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor's choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. Source Code License.

2.1. The Initial Developer Grant.
The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work; and

(b) under Patents Claims infringed by the making, using or selling of Original Code, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Code (or portions thereof).

(c) the licenses granted in this Section 2.1(a) and (b) are effective on the date Initial Developer first distributes Original Code under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: 1) for code that You delete from the Original Code; 2) separate from the Original Code; or 3) for infringements caused by: i) the modification of the Original Code or ii) the combination of the Original Code with other software or devices.

2.2. Contributor Grant.
Subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor, to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: 1) Modifications made by that Contributor (or portions thereof); and 2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) the licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first makes Commercial Use of the Covered Code.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: 1) for any code that Contributor has deleted from the Contributor Version; 2) separate from the Contributor Version;

3) for infringements caused by: i) third party modifications of Contributor Version or ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or 4) under Patent Claims infringed by Covered Code in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Application of License.
The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients' rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

3.2. Availability of Source Code.
Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.

3.3. Description of Modifications.
You must cause all Covered Code to which You contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

3.4. Intellectual Property Matters
(a) Third Party Claims.
If Contributor has knowledge that a license under a third party's

intellectual property rights is required to exercise the rights granted by such Contributor under Sections 2.1 or 2.2, Contributor must include a text file with the Source Code distribution titled "LEGAL" which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If Contributor obtains such knowledge after the Modification is made available as described in Section 3.2, Contributor shall promptly modify the LEGAL file in all copies Contributor makes available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs.
If Contributor's Modifications include an application programming interface and Contributor has knowledge of patent licenses which are reasonably necessary to implement that API, Contributor must also include this information in the LEGAL file.

(c) Representations.
Contributor represents that, except as disclosed pursuant to Section 3.4(a) above, Contributor believes that Contributor's Modifications are Contributor's original creation(s) and/or Contributor has sufficient rights to grant the rights conveyed by this License.

3.5. Required Notices.
You must duplicate the notice in Exhibit A in each file of the Source Code. If it is not possible to put such notice in a particular Source Code file due to its structure, then You must include such notice in a location (such as a relevant directory) where a user would be likely to look for such a notice. If You created one or more Modification(s) You may add your name as a Contributor to the notice described in Exhibit A. You must also duplicate this License in any documentation for the Source Code where You describe recipients' rights or ownership rights relating to Covered Code. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. Distribution of Executable Versions.
You may distribute Covered Code in Executable form only if the

requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code or ownership rights under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.7. Larger Works.
You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Application of this License.

This License applies to code to which the Initial Developer has attached the notice in Exhibit A and to related Covered Code.

6. Versions of the License.

6.1. New Versions.

Netscape Communications Corporation ("Netscape") may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.

6.2. Effect of New Versions.
Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Netscape. No one other than Netscape has the right to modify the terms applicable to Covered Code created under this License.

6.3. Derivative Works.
If You create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), You must (a) rename Your license so that the phrases "Mozilla", "MOZILLAPL", "MOZPL", "Netscape", "MPL", "NPL" or any confusingly similar phrase do not appear in your license (except to note that your license differs from this License) and (b) otherwise make it clear that Your version of the license contains terms which differ from the Mozilla Public License and Netscape Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. TERMINATION.

8.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

8.2. If You initiate litigation by asserting a patent infringement claim (excluding declatory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You file such action is referred to as "Participant") alleging that:

(a) such Participant's Contributor Version directly or indirectly infringes any patent, then any and all rights granted by such Participant to You under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively, unless if within 60 days after receipt of notice You either: (i) agree in writing to pay Participant a mutually agreeable reasonable royalty for Your past and future use of Modifications made by such Participant, or (ii) withdraw Your litigation claim with respect to the Contributor Version against such Participant. If within 60 days of notice, a reasonable royalty and payment arrangement are not mutually agreed upon in writing by the parties or the litigation claim is not withdrawn, the rights granted by Participant to You under Sections 2.1 and/or 2.2 automatically terminate at the expiration of the 60 day notice period specified above.

(b) any software, hardware, or device, other than such Participant's Contributor Version, directly or indirectly infringes any patent, then any rights granted to You by such Participant under Sections 2.1(b) and 2.2(b) are revoked effective as of the date You first made, used, sold, distributed, or had made, Modifications made by that Participant.

8.3. If You assert a patent infringement claim against Participant alleging that such Participant's Contributor Version directly or indirectly infringes any patent where such claim is resolved (such as by license or settlement) prior to the initiation of patent infringement litigation, then the reasonable value of the licenses granted by such Participant under Sections 2.1 or 2.2 shall be taken into account in determining the amount or value of any payment or license.

8.4. In the event of termination under Sections 8.1 or 8.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or any distributor hereunder prior to termination shall survive termination.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR

ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein.

11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in the United States of America, any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such

responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

13. MULTIPLE-LICENSED CODE.

Initial Developer may designate portions of the Covered Code as "Multiple-Licensed". "Multiple-Licensed" means that the Initial Developer permits you to utilize portions of the Covered Code under Your choice of the NPL or the alternative licenses, if any, specified by the Initial Developer in the file described in Exhibit A.

EXHIBIT A -Mozilla Public License.

``The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is _____.

The Initial Developer of the Original Code is _____. Portions created by _____ are Copyright (C) _____ _____. All Rights Reserved.

Contributor(s): _____.

Alternatively, the contents of this file may be used under the terms of the _____ license (the "[___] License"), in which case the provisions of [_____] License are applicable instead of those above. If you wish to allow use of your version of this file only under the terms of the [____] License and not to allow others to use your version of this file under the MPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the [___] License. If you do not delete the provisions above, a recipient may use your version of this file under either the MPL or the [___] License."

[NOTE: The text of this Exhibit A may differ slightly from the text of the notices in the Source Code files of the Original Code. You should use the text of this Exhibit A rather than the text found in the Original Code Source Code for Your Modifications.]

1.60 cxf-rt-rs-security-cors 2.5.3

1.60.1 Available under license :

Apache CXF
Copyright 2006-2012 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work

(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses

granted to You under this License for that Work shall terminate
as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]"

replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.61 dom4j 2.1.1

1.62 esapi 2.1.0

1.62.1 Available under license :

/**
* OWASP Enterprise Security API (ESAPI)
*
* This file is part of the Open Web Application Security Project (OWASP)
* Enterprise Security API (ESAPI) project. For details, please see
* http://www.owasp.org/index.php/ESAPI.
*
* Copyright (c) 2007 - The OWASP Foundation
*
* The ESAPI is published by OWASP under the BSD license. You should read and accept the
* LICENSE before you use, modify, and/or redistribute this software.
*
* @author Jeff Williams Aspect Security
* @created 2007
*/
The BSD 2-Clause License

The following is a BSD 2-Clause license template. To generate your own license, change the values of OWNER and YEAR from their original values as given here, and substitute your own.

Note: see also the BSD-3-Clause license.

This prelude is not part of the license.

<OWNER> = Regents of the University of California
<YEAR> = 1998

In the original BSD license, both occurrences of the phrase "COPYRIGHT HOLDERS AND CONTRIBUTORS" in the disclaimer read "REGENTS AND CONTRIBUTORS".

Here is the license template:

Copyright (c) <YEAR>, <OWNER>
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This project and all associated code is Copyright (c) 2007 - The OWASP Foundation

This project licensed under the BSD license, which is very permissive and about as close to public domain as is possible. You can use or modify ESAPI however you want, even include it in commercial products.

1.63 geronimo-jms_1.1_spec 1.1.1

1.63.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain

separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the

origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache Geronimo
Copyright 2003-2008 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.64 geronimo-stax-api 1.0.1 :1.0

1.64.1 Available under license :

Apache License, Version 2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and

2. You must cause any modified files to carry prominent notices stating that You changed the files; and

3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend,

and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.
Apache Geronimo
Copyright 2003-2006 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

1.65 geronimo-stax-api_1.0_spec 1.0.1 :wso2v2

1.65.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity

exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided

that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,

or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

```
/*
**
** Licensed to the Apache Software Foundation (ASF) under one
** or more contributor license agreements.  See the NOTICE file
** distributed with this work for additional information
** regarding copyright ownership.  The ASF licenses this file
** to you under the Apache License, Version 2.0 (the
** "License"); you may not use this file except in compliance
** with the License.  You may obtain a copy of the License at
**
**  http://www.apache.org/licenses/LICENSE-2.0
**
** Unless required by applicable law or agreed to in writing,
** software distributed under the License is distributed on an
```

** "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
** KIND, either express or implied. See the License for the
** specific language governing permissions and limitations
** under the License.
*/

1.66 geronimo-ws-metadata_2.0_spec 1.1.3

1.66.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work

(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses

granted to You under this License for that Work shall terminate
as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]"

replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Web Services Metadata 2.0
Copyright 2003-2010 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

1.67 Gson 2.3.1

1.67.1 Available under license :

```
/*
* Copyright (C) 2008 Google Inc.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.google.gson.internal;
```

```
import java.lang.reflect.Type;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

/**
* Contains static utility methods pertaining to primitive types and their
* corresponding wrapper types.
*
* @author Kevin Bourrillion
*/
public final class Primitives {
 private Primitives() {}

 /** A map from primitive types to their corresponding wrapper types. */
 private static final Map<Class<?>, Class<?>> PRIMITIVE_TO_WRAPPER_TYPE;

 /** A map from wrapper types to their corresponding primitive types. */
 private static final Map<Class<?>, Class<?>> WRAPPER_TO_PRIMITIVE_TYPE;

 // Sad that we can't use a BiMap. :(

 static {
   Map<Class<?>, Class<?>> primToWrap = new HashMap<Class<?>, Class<?>>(16);
   Map<Class<?>, Class<?>> wrapToPrim = new HashMap<Class<?>, Class<?>>(16);

   add(primToWrap, wrapToPrim, boolean.class, Boolean.class);
   add(primToWrap, wrapToPrim, byte.class, Byte.class);
   add(primToWrap, wrapToPrim, char.class, Character.class);
   add(primToWrap, wrapToPrim, double.class, Double.class);
   add(primToWrap, wrapToPrim, float.class, Float.class);
   add(primToWrap, wrapToPrim, int.class, Integer.class);
   add(primToWrap, wrapToPrim, long.class, Long.class);
   add(primToWrap, wrapToPrim, short.class, Short.class);
   add(primToWrap, wrapToPrim, void.class, Void.class);

   PRIMITIVE_TO_WRAPPER_TYPE = Collections.unmodifiableMap(primToWrap);
   WRAPPER_TO_PRIMITIVE_TYPE = Collections.unmodifiableMap(wrapToPrim);
 }

 private static void add(Map<Class<?>, Class<?>> forward,
     Map<Class<?>, Class<?>> backward, Class<?> key, Class<?> value) {
   forward.put(key, value);
   backward.put(value, key);
 }
```

```
/**
 * Returns true if this type is a primitive.
 */
public static boolean isPrimitive(Type type) {
  return PRIMITIVE_TO_WRAPPER_TYPE.containsKey(type);
}

/**
 * Returns {@code true} if {@code type} is one of the nine
 * primitive-wrapper types, such as {@link Integer}.
 *
 * @see Class#isPrimitive
 */
public static boolean isWrapperType(Type type) {
  return WRAPPER_TO_PRIMITIVE_TYPE.containsKey(
      $Gson$Preconditions.checkNotNull(type));
}

/**
 * Returns the corresponding wrapper type of {@code type} if it is a primitive
 * type; otherwise returns {@code type} itself. Idempotent.
 * <pre>
 *     wrap(int.class) == Integer.class
 *     wrap(Integer.class) == Integer.class
 *     wrap(String.class) == String.class
 * </pre>
 */
public static <T> Class<T> wrap(Class<T> type) {
  // cast is safe: long.class and Long.class are both of type Class<Long>
  @SuppressWarnings("unchecked")
  Class<T> wrapped = (Class<T>) PRIMITIVE_TO_WRAPPER_TYPE.get(
      $Gson$Preconditions.checkNotNull(type));
  return (wrapped == null) ? type : wrapped;
}

/**
 * Returns the corresponding primitive type of {@code type} if it is a
 * wrapper type; otherwise returns {@code type} itself. Idempotent.
 * <pre>
 *     unwrap(Integer.class) == int.class
 *     unwrap(int.class) == int.class
 *     unwrap(String.class) == String.class
 * </pre>
 */
public static <T> Class<T> unwrap(Class<T> type) {
  // cast is safe: long.class and Long.class are both of type Class<Long>
  @SuppressWarnings("unchecked")
  Class<T> unwrapped = (Class<T>) WRAPPER_TO_PRIMITIVE_TYPE.get(
```

```
      $Gson$Preconditions.checkNotNull(type));
  return (unwrapped == null) ? type : unwrapped;
 }
}
```

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the

Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of

any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and
limitations under the License.

1.68 guava 26.0-jre

1.69 Hamcrest Core 1.1

1.69.1 Available under license :

The BSD 3-Clause License

The following is a BSD 3-Clause ("BSD New" or "BSD Simplified") license template. To generate your own license, change the values of OWNER, ORGANIZATION and YEAR from their original values as given here, and substitute your own.

Note: You may omit clause 3 and still be OSD-conformant. Despite its colloquial name "BSD New", this is not the newest version of the BSD license; it was followed by the even newer BSD-2-Clause version, sometimes known as the "Simplified BSD License". On January 9th, 2008 the OSI Board approved BSD-2-Clause, which is used by FreeBSD and others. It omits the final "no-endorsement" clause and is thus roughly equivalent to the MIT License.

Historical Background: The original license used on BSD Unix had four clauses. The advertising clause (the third of four clauses) required you to acknowledge use of U.C. Berkeley code in your advertising of any product using that code. It was officially rescinded by the Director of the Office of Technology Licensing of the University of California on July 22nd, 1999. He states that clause 3 is "hereby deleted in its entirety." The four clause license has not been approved by OSI. The license below does not contain the advertising clause.

This prelude is not part of the license.

<OWNER> = Regents of the University of California
<ORGANIZATION> = University of California, Berkeley
<YEAR> = 1998

In the original BSD license, the occurrence of "copyright holder" in the 3rd clause read "ORGANIZATION", placeholder for "University of California". In the original BSD license, both occurrences of the phrase "COPYRIGHT HOLDERS AND CONTRIBUTORS" in the disclaimer read "REGENTS AND CONTRIBUTORS".

Here is the license template:

Copyright (c) <YEAR>, <OWNER>
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.70 hamcrest-core 1.1

1.70.1 Available under license :

BSD License

Copyright (c) 2000-2006, www.hamcrest.org
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of
conditions and the following disclaimer. Redistributions in binary form must reproduce
the above copyright notice, this list of conditions and the following disclaimer in
the documentation and/or other materials provided with the distribution.

Neither the name of Hamcrest nor the names of its contributors may be used to endorse
or promote products derived from this software without specific prior written
permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,

INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object

form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a

file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.71 hamcrest-core_test 1.1

1.71.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,

including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual,

worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf

of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.72 hawtbuf 1.11

1.73 hibernate-core 3.6.9

1.73.1 Available under license :

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original

author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that

you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it

contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application

to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact
that part of it is a work based on the Library, and explaining
where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any

patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR

OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software

Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the library, if
necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the
library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice

That's all there is to it!

1.74 hibernate-jpa-2.0-api 1.0.1.Final

1.74.1 Available under license :

Eclipse Distribution License - v 1.0

Copyright (c) 2007, Eclipse Foundation, Inc. and its licensors.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
Neither the name of the Eclipse Foundation, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.75 httpclient 4.5.5

1.75.1 Available under license :

Apache HttpClient
Copyright 1999-2018 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,

including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual,

worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf

of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.76 httpcore 4.4.9

1.76.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise

designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must

include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly

negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache HttpCore

1.77 jackson-core-asl 1.9.13

1.77.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate

as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify

the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include

the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
/* Jackson JSON-processor.
*
* Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi
*
* Licensed under the License specified in file LICENSE, included with
* the source code and binary code bundles.
* You may not use this file except in compliance with the License.
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]

Preamble

The licenses for most software are designed to take away your

freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be

consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The

former contains code derived from the library, whereas the latter must
be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other
program which contains a notice placed by the copyright holder or
other authorized party saying it may be distributed under the terms of
this Lesser General Public License (also called "this License").
Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data
prepared so as to be conveniently linked with application programs
(which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work
which has been distributed under these terms. A "work based on the
Library" means either the Library or any derivative work under
copyright law: that is to say, a work containing the Library or a
portion of it, either verbatim or with modifications and/or translated
straightforwardly into another language. (Hereinafter, translation is
included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for
making modifications to it. For a library, complete source code means
all the source code for all modules it contains, plus any associated
interface definition files, plus the scripts used to control compilation
and installation of the library.

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope. The act of
running a program using the Library is not restricted, and output from
such a program is covered only if its contents constitute a work based
on the Library (independent of the use of the Library in a tool for
writing it). Whether that is true depends on what the Library does
and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's
complete source code as you receive it, in any medium, provided that
you conspicuously and appropriately publish on each copy an
appropriate copyright notice and disclaimer of warranty; keep intact
all the notices that refer to this License and to the absence of any
warranty; and distribute a copy of this License along with the
Library.

You may charge a fee for the physical act of transferring a copy,
and you may at your option offer warranty protection in exchange for a

fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of

a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be

linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is

interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at
least three years, to give the same user the materials
specified in Subsection 6a, above, for a charge no more
than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy
from a designated place, offer equivalent access to copy the above
specified materials from the same place.

e) Verify that the user has already received a copy of these
materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the
Library" must include any data and utility programs needed for
reproducing the executable from it. However, as a special exception,
the materials to be distributed need not include anything that is
normally distributed (in either source or binary form) with the major
components (compiler, kernel, and so on) of the operating system on
which the executable runs, unless that component itself accompanies
the executable.

It may happen that this requirement contradicts the license
restrictions of other proprietary libraries that do not normally
accompany the operating system. Such a contradiction means you cannot
use both them and the Library together in an executable that you
distribute.

7. You may place library facilities that are a work based on the
Library side-by-side in a single library together with other library
facilities not covered by this License, and distribute such a combined
library, provided that the separate distribution of the work based on
the Library and of the other library facilities is otherwise
permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work
based on the Library, uncombined with any other library
facilities. This must be distributed under the terms of the
Sections above.

b) Give prominent notice with the combined library of the fact
that part of it is a work based on the Library, and explaining
where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute
the Library except as expressly provided under this License. Any
attempt otherwise to copy, modify, sublicense, link with, or

distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot

impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Library under this License may add
an explicit geographical distribution limitation excluding those countries,
so that distribution is permitted only in or among countries not thus
excluded. In such case, this License incorporates the limitation as if
written in the body of this License.

13. The Free Software Foundation may publish revised and/or new
versions of the Lesser General Public License from time to time.
Such new versions will be similar in spirit to the present version,
but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library
specifies a version number of this License which applies to it and
"any later version", you have the option of following the terms and
conditions either of that version or of any later version published by
the Free Software Foundation. If the Library does not specify a
license version number, you may choose any version ever published by
the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free
programs whose distribution conditions are incompatible with these,
write to the author to ask for permission. For software which is
copyrighted by the Free Software Foundation, write to the Free
Software Foundation; we sometimes make exceptions for this. Our
decision will be guided by the two goals of preserving the free status
of all derivatives of our free software and of promoting the sharing
and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO
WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW.
EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR
OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY
KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME
THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN

WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the
library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice

That's all there is to it!

1.78 jackson-core-lgpl 1.7.4

1.78.1 Available under license :

```
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Marker annotation that can be used to define a non-static,
* single-argument method to be used as a "setter" for a logical property
* as an alternative to recommended
* {@link JsonProperty} annotation (which was introduced in version 1.1).
*<p>
* Setter means that when a property with matching name is encountered in
* JSON content, this method will be used to set value of the property.
*<p>
* NOTE: this annotation was briefly deprecated for version 1.5; but has
* since been un-deprecated to both allow for asymmetric naming (possibly
* different name when reading and writing JSON), and more importantly to
* allow multi-argument setter method in future.
*/
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonSetter
{
  /**
   * Optional default argument that defines logical property this
   * method is used to modify ("set"); this is the property
   * name used in JSON content.
   */
  String value() default "";
}
```

```
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Annotation that can be used to define whether object properties
* that have null values are to be written out when serializing
* content as JSON. This affects Bean and Map serialization.
*<p>
* Annotation can be used with Classes (all instances of
* given class) and Methods.
*<p>
* Default value for this property is 'true', meaning that null
* properties are written.
*<p>
* @deprecated (since 1.6) Currently recommended annotation to use is
* {@link org.codehaus.jackson.map.annotate.JsonSerialize#include()}
* (with values <code>ALWAYS</code> or <code>NON_NULL</code>)
*/
@Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
@Deprecated
public @interface JsonWriteNullProperties
{
  /**
   * Whether properties for beans of annotated type will always be
   * written (true), or only if not null (false).
   */
  boolean value() default true;
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Annotation that can be used to mark "setter" methods to indicate the
* actual type of values contained in a container type that is value
* of the property associated with the method.
* (phew! that's a mouthful!).
* This is usually done if the declared element type is abstract or
* too generic; annotation can denote actual concrete type to
```

```
* instantiate when deserializing contents of the container.
* To define type of the actual container itself, use
* {@link JsonClass} instead.
*<p>
* Note that the indicated type must be compatible with the declared
* type; that is, it has to be a sub-type or implementation of
* the declared type. This is usually the case; and if it wasn't
* then the call to associated "setter" method would fail with
* a type-mismatch exception.
*
* @deprecated As of version 1.1, use {@link org.codehaus.jackson.map.annotate.JsonDeserialize#contentAs}
instead
*/
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
@Deprecated
public @interface JsonContentClass
{
  /**
   * Class that is the expected concrete value type of the container
   * (which is value of the property associated
   * with the annotated method). Will be used by deserializer to
   * instantiate the type, using
   *<p>
   * Note: if a non-property method is annotated with this annotation,
   * deserializer will throw an exception to denote invalid annotation.
   */
  public Class<?> value();
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Marker annotation that can be used to define a non-static,
* single-argument method, to be used as a "fallback" handler
* for all otherwise unrecognized properties found from Json content.
* It is similar to {@link javax.xml.bind.annotation.XmlAnyElement}
* in behavior; and can only be used to denote a single property
* per type.
*<p>
* If used, all otherwise unmapped key-value pairs from Json Object
* structs are added to the property (of type Map or bean).
*/
```

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonAnySetter
{
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Annotation that can be used with "setter" methods to
* indicate the actual type to use for deserializing value
* of the associated logical property.
* This is usually done if the declared type is abstract or too generic;
* annotation can denote actual concrete type to instantiate when
* deserializing the property.
*<p>
* The indicated type must be compatible with the declared
* type. For deserialization (setters) this means that
* it has to be a sub-type or implementation of
* the declared type.
* If this constraint is violated, an exception (usually
* {@link IllegalArgumentException}) can be thrown by runtime.
*<p>
* Note that for container types (arrays, Lists/Collections/Maps) this
* indicates the type of container itself; for contained Objects, use
* {@link JsonContentClass} instead (or for Map keys,
* {@link JsonKeyClass}).
*
* @deprecated As of version 1.1, use {@link org.codehaus.jackson.map.annotate.JsonDeserialize#as} instead
*/
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
@Deprecated
public @interface JsonClass
{
  /**
   * Class that is the type to use for deserializating value of
   * the property associated
   * with the annotated method.
   */
  public Class<?> value();
}
```

```
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Marker annotation that can be used to define a non-static
* method as a "setter" or "getter" for a logical property
* (depending on its signature),
* or non-static object field to be used (serialized, deserialized) as
* a logical property.
*<p>
* Default value ("") indicates that the field name is used
* as the property name without any modifications, but it
* can be specified to non-empty value to specify different
* name. Property name refers to name used externally, as
* the field name in Json objects.
*<p>
* NOTE: since version 1.1, annotation has also been applicable
* to fields (not with 1.0).
*<p>
* NOTE: since version 1.2, annotation has also been applicable
* to (constructor) parameters
*/
@Target({ElementType.FIELD, ElementType.METHOD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonProperty
{
  /**
   * Defines name of the logical property, i.e. Json object field
   * name to use for the property: if empty String (which is the
   * default), will use name of the field that is annotated.
   */
  String value() default "";
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Annotation that can be used to mark "setter" methods to indicate the
* actual type of key Objects for the Map type that is value
```

```
* of the property associated with the method.
* This is usually done if the declared element type is abstract or
* too generic; annotation can denote actual concrete type to
* instantiate when deserializing contents of the container.
* To define type of the actual container itself, use
* {@link JsonClass} instead.
*<p>
* Note that the indicated type must be compatible with the declared
* type; that is, it has to be a sub-type or implementation of
* the declared type. This is usually the case; and if it wasn't
* then the call to associated "setter" method would fail with
* a type-mismatch exception.
*<p>
* Note: while any class can be indicated as the Key class, there
* must be a registered Key Deserializer for the type.
*
* @deprecated As of version 1.1, use {@link org.codehaus.jackson.map.annotate.JsonDeserialize#keyAs} instead
*/
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
@Deprecated
public @interface JsonKeyClass
{
  /**
   * Class that is the expected concrete value type of the container
   * (which is value of the property associated
   * with the annotated method). Will be used by deserializer to
   * instantiate the type, using
   *<p>
   * Note: if a non-property method is annotated with this annotation,
   * deserializer will throw an exception to denote invalid annotation.
   */
  public Class<?> value();
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Meta-annotation (annotations used on other annotations)
* used for marking all annotations that are
* part of Jackson package. Can be used for recognizing all
* Jackson annotations generically, and in future also for
* passing other generic annotation configuration.
```

```
*/
@Target({ElementType.ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface JacksonAnnotation
{
  // for now, a pure tag annotation, no parameters
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Annotation used with {@link JsonTypeInfo} to indicate sub types of serializable
* polymorphic types, and to associate logical names used within JSON content
* (which is more portable than using physical Java class names).
*
* @since 1.5
*/
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonSubTypes {
  /**
   * Subtypes of the annotated type (annotated class, or property value type
   * associated with the annotated method). These will be checked recursively
   * so that types can be defined by only including direct subtypes.
   */
  public Type[] value();

  /**
   * Definition of a subtype, along with optional name. If name is missing, class
   * of the type will be checked for {@link JsonTypeName} annotation; and if that
   * is also missing or empty, a default
   * name will be constructed by type id mechanism.
   * Default name is usually based on class name.
   */
  public @interface Type {
     /**
      * Class of the subtype
      */
     public Class<?> value();

     /**
      * Logical type name used as the type identifier for the class
      */
```

```
    public String name() default "";
  }
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;


/**
* Annotation used for binding logical name that the annotated class
* has. Used with {@link JsonTypeInfo} (and specifically its
* {@link JsonTypeInfo#use} property) to establish relationship
* between type names and types.
*
* @since 1.5
*
* @author tatu
*/
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonTypeName {
  /**
   * Logical type name for annotated type. If missing (or defined as Empty String),
   * defaults to using non-qualified class name as the type.
   */
  public String value() default "";
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Annotation that can be used to define ordering (possibly partial) to use
* when serializing object properties. Properties included in annotation
* declaration will be serialized first (in defined order), followed by
* any properties not included in the definition.
* Annotation definition will override any implicit orderings (such as
* guarantee that Creator-properties are serialized before non-creator
* properties)
*<p>
* Examples:
```

```
*<pre>
*  // ensure that "id" and "name" are output before other properties
*  <div>@</div>JsonPropertyOrder({ "id", "name" })
*  // order any properties that don't have explicit setting using alphabetic order
*  <div>@</div>JsonPropertyOrder(alphabetic=true)
*</pre>
*<p>
* This annotation has no effect on deserialization.
*
* @since 1.4
*/
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonPropertyOrder
{
  /**
   * Order in which properties of annotated object are to be serialized in.
   */
  public String[] value() default { };

  /**
   * Property that defines what to do regarding ordering of properties
   * not explicitly included in annotation instance. If set to true,
   * they will be alphabetically ordered; if false, order is
   * undefined (default setting)
   */
  public boolean alphabetic() default false;
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Marker annotation that indicates that all properties of annotated
* type are to be ignored during serialization and deserialization.
*<p>
* Note: annotation does have boolean 'value' property (which defaults
* to 'true'), so that it is actually possible to override value
* using mix-in annotations.
*
* @since 1.7
*/
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
```

```
@JacksonAnnotation
public @interface JsonIgnoreType
{
  /**
   * Optional argument that defines whether this annotation is active
   * or not. The only use for value 'false' if for overriding purposes
   * (which is not needed often); most likely it is needed for use
   * with "mix-in annotations" ("annotation overrides").
   * For most cases, however, default value of "true" is just fine
   * and should be omitted.
   */
  boolean value() default true;

}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Marker annotation similar to
* {@link javax.xml.bind.annotation.XmlValue}
* that indicates that results of the annotated "getter" method
* (which means signature must be that of getters; non-void return
* type, no args) is to be used as the single value to serialize
* for the instance. Usually value will be of a simple scalar type
* (String or Number), but it can be any serializable type (Collection,
* Map or Bean).
*<p>
* At most one method of a Class can be annotated with this annotation;
* if more than one is found, an exception may be thrown.
* Also, if method signature is not compatible with Getters, an exception
* may be thrown.
* Whether exception is thrown or not is an implementation detail (due
* to filtering during introspection, some annotations may be skipped)
* and applications should not rely on specific behavior.
*<p>
* A typical use case is that of annotating <code>toString()</code>
* method so that returned String value is Object's Json serialization.
*<p>
* Boolean argument is only used so that sub-classes can "disable"
* annotation if necessary.
*/
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
```

```
public @interface JsonValue
{
  /**
   * Optional argument that defines whether this annotation is active
   * or not. The only use for value 'false' if for overriding purposes.
   * Overriding may be necessary when used
   * with "mix-in annotations" (aka "annotation overrides").
   * For most cases, however, default value of "true" is just fine
   * and should be omitted.
   */
  boolean value() default true;
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Marker annotation that indicates that the annotated method
* or field should be serialized by including literal String value
* of the property as is, without quoting of characters.
* This can be useful for injecting values already serialized in JSON or
* passing javascript function definitions from server to a javascript client.
*<p>
* Warning: the resulting JSON stream may be invalid depending on your input value.
*
* @since 1.7.0
*/
@Target( { ElementType.METHOD, ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonRawValue
{
  /**
   * Optional argument that defines whether this annotation is active
   * or not. The only use for value 'false' if for overriding purposes
   * (which is not needed often); most likely it is needed for use
   * with "mix-in annotations" (aka "annotation overrides").
   * For most cases, however, default value of "true" is just fine
   * and should be omitted.
   */
  boolean value() default true;
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
```

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Annotation used to indicate that annotated property is part of
* two-way linkage between fields; and that its role is "parent" (or "forward") link.
* Value type (class) of property must have a single compatible property annotated with
* {@link JsonBackReference}. Linkage is handled such that the property
* annotated with this annotation is handled normally (serialized normally, no
* special handling for deserialization); it is the matching back reference
* that requires special handling
*<p>
* All references have logical name to allow handling multiple linkages; typical case
* would be that where nodes have both parent/child and sibling linkages. If so,
* pairs of references should be named differently.
* It is an error for a class too have multiple managed references with same name,
* even if types pointed are different.
*<p>
* Note: only methods and fields can be annotated with this annotation: constructor
* arguments should NOT be annotated, as they can not be either managed or back
* references.
*
* @author tatu
*/
@Target({ElementType.FIELD, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonManagedReference
{
  /**
   * Logical have for the reference property pair; used to link managed and
   * back references. Default name can be used if there is just single
   * reference pair (for example, node class that just has parent/child linkage,
   * consisting of one managed reference and matching back reference)
   */
  public String value() default "defaultReference";
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Annotation used for configuring details of if and how type information is
* used with JSON serialization and deserialization, to preserve information
```

```
* about actual class of Object instances. This is necessarily for polymorphic
* types, and may also be needed to link abstract declared types and matching
* concrete implementation.
*<p>
* Some examples of typical annotations:
*<pre>
*  // Include Java class name ("com.myempl.ImplClass") as JSON property "class"
*  \@JsonTypeInfo(use=Id.CLASS, include=As.PROPERTY, property="class")
*
*  // Include logical type name (defined in impl classes) as wrapper; 2 annotations
*  \@JsonTypeInfo(use=Id.NAME, include=As.WRAPPER_OBJECT)
*  \@JsonSubTypes({com.myemp.Impl1.class, com.myempl.Impl2.class})
*</pre>
* Alternatively you can also define fully customized type handling by using
* {@link org.codehaus.jackson.map.annotate.JsonTypeResolver} annotation.
*<p>
* NOTE: originally this annotation was only available to use with types (classes),
* but starting with 1.7, it is also allowed for properties (fields, methods,
* constructor parameters).
*<p>
* When used for properties (fields, methods), there annotation always defines
* to <b>values</b>: specifically, when applied to a <code>Collection</code> or
* <code>Map</code> property, it will <b>not</b> apply to container type but
* to contained values. This is identical to how JAXB handles type information
* annotations; and is chosen since it is the dominant use case. There is no
* per-property way to force type information to be included for type of
* container itself.
*
* @see org.codehaus.jackson.map.annotate.JsonTypeResolver
* @since 1.5 (but available to fields, methods and constructor params since 1.7)
*
* @author tatu
*/
@Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonTypeInfo
{
  /*
  /**********************************************************
  /* Value enumerations used for properties
  /**********************************************************
   */

  /**
   * Definition of different type identifiers that can be included in JSON
   * during serialization, and used for deserialization.
   */
```

```
public enum Id {
    /**
     * This means that no explicit type metadata is included, and typing is
     * purely done using contextual information possibly augmented with other
     * annotations.
     *<p>
     * Note: no {@link org.codehaus.jackson.map.jsontype.TypeIdResolver}
     * is constructed if this value is used.
     */
    NONE(null),

    /**
     * Means that fully-qualified Java class name is used as the type identifier.
     */
    CLASS("@class"),

    /**
     * Means that Java class name with minimal path is used as the type identifier.
     * Minimal means that only class name, and that part of preceding Java
     * package name is included that is needed to construct fully-qualified name
     * given fully-qualified name of the declared supertype.
     * For example, for supertype "com.foobar.Base", and concrete type
     * "com.foo.Impl", only "Impl" would be included; and for "com.foo.impl.Impl2"
     * only "impl.Impl2" would be included.
     *<p>
     * If all related classes are in the same Java package, this option can reduce
     * amount of type information overhead, especially for small types.
     * However, please note that using this alternative is inherently risky since it
     * assumes that the
     * supertype can be reliably detected. Given that it is based on declared type
     * (since ultimate supertype, <code>java.lang.Object</code> would not be very
     * useful reference point), this may not always work as expected.
     */
    MINIMAL_CLASS("@c"),

    /**
     * Means that logical type name is used as type information; name will then need
     * to be separately resolved to actual concrete type (Class).
     */
    NAME("@type"),

    /**
     * Means that typing mechanism uses customized handling, with possibly
     * custom configuration. This means that semantics of other properties is
     * not defined by Jackson package, but by the custom implementation.
     */
    CUSTOM(null)
    ;
```

```
    private final String _defaultPropertyName;

    private Id(String defProp) {
        _defaultPropertyName = defProp;
    }

    public String getDefaultPropertyName() { return _defaultPropertyName; }
}

/**
 * Definition of standard type inclusion mechanisms for type metadata.
 * Used for standard metadata types, except for {@link Id#NONE}.
 * May or may not be used for custom types ({@link Id#CUSTOM}).
 */
public enum As {
    /**
     * Inclusion mechanism that uses a single configurable property, included
     * along with actual data (POJO properties) as a separate meta-property.
     * <p>
     * Default choice for inclusion.
     */
    PROPERTY,

    /**
     * Inclusion mechanism that wraps typed JSON value (POJO
     * serialized as JSON) in
     * a JSON Object that has a single entry,
     * where field name is serialized type identifier,
     * and value is the actual JSON value.
     *<p>
     * Note: can only be used if type information can be serialized as
     * String. This is true for standard type metadata types, but not
     * necessarily for custom types.
     */
    WRAPPER_OBJECT,

    /**
     * Inclusion mechanism that wraps typed JSON value (POJO
     * serialized as JSON) in
     * a 2-element JSON array: first element is the serialized
     * type identifier, and second element the serialized POJO
     * as JSON Object.
     */
    WRAPPER_ARRAY,
    ;
}
```

```
  /*
  /**********************************************************
  /* Annotation properties
  /**********************************************************
   */

  /**
   * What kind of type metadata is to be used for serializing and deserializing
   * type information for instances of annotated type (and its subtypes
   * unless overridden)
   */
  public Id use();

  /**
   * What mechanism is used for including type metadata (if any; for
   * {@link Id#NONE} nothing is included). Default
   *<p>
   * Note that for type metadata type of {@link Id#CUSTOM},
   * this setting may or may not have any effect.
   */
  public As include() default As.PROPERTY;

  /**
   * Property names used when type inclusion method ({@link As#PROPERTY}) is used
   * (or possibly when using type metadata of type {@link Id#CUSTOM}).
   *<p>
   * Default property name used if this property is not explicitly defined
   * (or is set to empty String) is based on
   * type metadata type ({@link #use}) used.
   */
  public String property() default "";
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Marker annotation that can be used to define a non-static,
* no-argument value-returning (non-void) method to be used as a "getter"
* for a logical property,
* as an alternative to recommended
* {@link JsonProperty} annotation (which was introduced in version 1.1).
*<p>
* Getter means that when serializing Object instance of class that has
* this method (possibly inherited from a super class), a call is made
```

```
* through the method, and return value will be serialized as value of
* the property.
*
* @deprecated Use {@link JsonProperty} instead (deprecated since version 1.5)
*/
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
@Deprecated
public @interface JsonGetter
{
  /**
   * Defines name of the logical property this
   * method is used to access ("get"); empty String means that
   * name should be derived from the underlying method (using
   * standard Bean name detection rules)
   */
  String value() default "";
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Marker annotation that can be used to define a non-static,
* no-argument method or member field as something of a reverse of
* {@link JsonAnySetter} method; basically being used like a
* getter but such that contents of the returned Map (type <b>must</b> be
* {@link java.util.Map}) are serialized as if they were actual properties
* of the bean that contains method/field with this annotations.
* As with {@link JsonAnySetter}, only one property should be annotated
* with this annotation.
*
* @since 1.6
*/
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonAnyGetter
{
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
```

```
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Marker annotation that can be used to define constructors and factory
* methods as one to use for instantiating new instances of the associated
* class.
*/
@Target({ElementType.METHOD, ElementType.CONSTRUCTOR})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonCreator
{
  // no values, since there's no property
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Annotation used to indicate that associated property is part of
* two-way linkage between fields; and that its role is "child" (or "back") link.
* Value type of the property must be a bean: it can not be a Collection, Map,
* Array or enumeration.
* Linkage is handled such that the property
* annotated with this annotation is not serialized; and during deserialization,
* its value is set to instance that has the "managed" (forward) link.
*<p>
* All references have logical name to allow handling multiple linkages; typical case
* would be that where nodes have both parent/child and sibling linkages. If so,
* pairs of references should be named differently.
* It is an error for a class to have multiple back references with same name,
* even if types pointed are different.
*<p>
* Note: only methods and fields can be annotated with this annotation: constructor
* arguments should NOT be annotated, as they can not be either managed or back
* references.
*
* @author tatu
*/
@Target({ElementType.FIELD, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonBackReference
{
```

```
  /**
   * Logical have for the reference property pair; used to link managed and
   * back references. Default name can be used if there is just single
   * reference pair (for example, node class that just has parent/child linkage,
   * consisting of one managed reference and matching back reference)
   */
  public String value() default "defaultReference";
}
package org.codehaus.jackson.annotate;

/**
* Enumeration used to define kinds of methods that annotations like
* {@link JsonAutoDetect} apply to.
*<p>
* In addition to actual method types (GETTER, SETTER, CREATOR; and
* sort-of-method, FIELD), 2 pseudo-types
* are defined for convenience: <code>ALWAYS</code> and <code>NONE</code>. These
* can be used to indicate, all or none of available method types (respectively),
* for use by annotations that takes <code>JsonMethod</code> argument.
*/
public enum JsonMethod
{
  /**
   * Getters are methods used to get a POJO field value for serialization,
   * or, under certain conditions also for de-serialization. Latter
   * can be used for effectively setting Collection or Map values
   * in absence of setters, iff returned value is not a copy but
   * actual value of the logical property.
   *<p>
   * Since version 1.3, this does <b>NOT</b> include "is getters" (methods
   * that return boolean and named 'isXxx' for property 'xxx'); instead,
   * {@link #IS_GETTER} is used}.
   */
  GETTER,

  /**
   * Setters are methods used to set a POJO value for deserialization.
   */
  SETTER,

     /**
      * Creators are constructors and (static) factory methods used to
      * construct POJO instances for deserialization
      */
     CREATOR,

     /**
      * Field refers to fields of regular Java objects. Although
```

```
     * they are not really methods, addition of optional field-discovery
     * in version 1.1 meant that there was need to enable/disable
     * their auto-detection, and this is the place to add it in.
     *
     * @since 1.1
     */
    FIELD,

    /**
     * "Is getters" are getter-like methods that are named "isXxx"
     * (instead of "getXxx" for getters) and return boolean value
     * (either primitive, or {@link java.lang.Boolean}).
     *
     * @since 1.3
     */
    IS_GETTER,

    /**
     * This pseudo-type indicates that none of real types is included
     */
    NONE,

    /**
     * This pseudo-type indicates that all of real types are included
     */
    ALL
    ;

private JsonMethod() { }

public boolean creatorEnabled() {
    return (this == CREATOR) || (this == ALL);
}

public boolean getterEnabled() {
    return (this == GETTER) || (this == ALL);
}

public boolean isGetterEnabled() {
    return (this == IS_GETTER) || (this == ALL);
}

public boolean setterEnabled() {
    return (this == SETTER) || (this == ALL);
}

public boolean fieldEnabled() {
    return (this == FIELD) || (this == ALL);
```

```
  }
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Marker annotation
* that indicates that the annotated method or field is to be ignored by
* introspection-based
* serialization and deserialization functionality. That is, it should
* not be consider a "getter", "setter" or "creator".
* <p>
* For example,
* a "getter" method that would otherwise denote
* a property (like, say, "getValue" to suggest property "value")
* to serialize, would be ignored and no such property would
* be output unless another annotation defines alternative method
* to use.
*<p>
* This annotation works purely on method-by-method (or field-by-field) basis;
* annotation on one method or field does not imply ignoring other methods
* or fields.
* Specifically, marking a "setter" candidate does not change handling
* of matching "getter" method (or vice versa).
*<p>
* Annotation is usually used just a like a marker annotation, that
* is, without explicitly defining 'value' argument (which defaults
* to <code>true</code>): but argument can be explicitly defined.
* This can be done to override an existing JsonIgnore by explictly
* defining one with 'false' argument.
*<p>
* Annotation is similar to {@link javax.xml.bind.annotation.XmlTransient}
*/
@Target({ElementType.METHOD, ElementType.CONSTRUCTOR, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonIgnore
{
  /**
   * Optional argument that defines whether this annotation is active
   * or not. The only use for value 'false' if for overriding purposes
   * (which is not needed often); most likely it is needed for use
   * with "mix-in annotations" (aka "annotation overrides").
   * For most cases, however, default value of "true" is just fine
```

```
   * and should be omitted.
   */
  boolean value() default true;
}
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.reflect.Member;
import java.lang.reflect.Modifier;

/**
* Class annotation that can be used to define which kinds of Methods
* are to be detected by auto-detection.
* Auto-detection means using name conventions
* and/or signature templates to find methods to use for data binding.
* For example, so-called "getters" can be auto-detected by looking for
* public member methods that return a value, do not take argument,
* and have prefix "get" in their name.
*<p>
* Pseudo-value <code>NONE</code> means that all auto-detection is disabled
* for the <b>specific</b> class that annotation is applied to (including
* its super-types, but only when resolving that class).
* Pseudo-value <code>ALWAYS</code> means that auto-detection is enabled
* for all method types for the class in similar way.
*<p>
* The default value is <code>ALWAYS</code>: that is, by default, auto-detection
* is enabled for all classes unless instructed otherwise.
*<p>
* Starting with version 1.5, it is also possible to use more fine-grained
* definitions, to basically define minimum visibility level needed. Defaults
* are different for different types (getters need to be public; setters can
* have any access modifier, for example).
*/
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonAutoDetect
{
 /**
  * Enumeration for possible visibility thresholds (minimum visibility)
  * that can be used to limit which methods (and fields) are
  * auto-detected.
  *
  * @since 1.5
  */
```

```
public enum Visibility {
 /**
  * Value that means that all kinds of access modifiers are acceptable,
  * from private to public.
  */
 ANY,
 /**
  * Value that means that any other access modifier other than 'private'
  * is considered auto-detectable.
  */
 NON_PRIVATE,
 /**
  * Value that means access modifiers 'protected' and 'public' are
  * auto-detectable (and 'private' and "package access" == no modifiers
  * are not)
  */
 PROTECTED_AND_PUBLIC,
 /**
  * Value to indicate that only 'public' access modifier is considered
  * auto-detectable.
  */
 PUBLIC_ONLY,
 /**
  * Value that indicates that no access modifiers are auto-detectable:
  * this can be used to explicitly disable auto-detection for specified
  * types.
  */
 NONE,

 /**
  * Value that indicates that default visibility level (whatever it is,
  * depends on context) is to be used. This usually means that inherited
  * value (from parent visibility settings) is to be used.
  */
 DEFAULT;

 public boolean isVisible(Member m) {
  switch (this) {
  case ANY:
     return true;
  case NONE:
     return false;
  case NON_PRIVATE:
     return !Modifier.isPrivate(m.getModifiers());
  case PROTECTED_AND_PUBLIC:
     if (Modifier.isProtected(m.getModifiers())) {
        return true;
     }
```

```
    // fall through to public case:
  case PUBLIC_ONLY:
    return Modifier.isPublic(m.getModifiers());
  }
  return false;
 }
}

 /**
  * Types of property elements (getters, setters, fields, creators) that
  * can be auto-detected.
  * NOTE: as of 1.5, it is recommended that instead of defining this property,
  * distinct visibility properties are used instead. This because levels
  * used with this method are not explicit, but global defaults that differ for different
  * methods. As such, this property can be considered <b>deprecated</b> and
  * only retained for backwards compatibility.
  */
 JsonMethod[] value() default { JsonMethod.ALL };

 /**
  * Minimum visibility required for auto-detecting regular getter methods.
  *
  * @since 1.5
  */
 Visibility getterVisibility() default Visibility.DEFAULT;

 /**
  * Minimum visibility required for auto-detecting is-getter methods.
  *
  * @since 1.5
  */
 Visibility isGetterVisibility() default Visibility.DEFAULT;

 /**
  * Minimum visibility required for auto-detecting setter methods.
  *
  * @since 1.5
  */
 Visibility setterVisibility() default Visibility.DEFAULT;

 /**
  * Minimum visibility required for auto-detecting Creator methods,
  * except for no-argument constructors (which are always detected
  * no matter what).
  *
  * @since 1.5
  */
 Visibility creatorVisibility() default Visibility.DEFAULT;
```

```
  /**
   * Minimum visibility required for auto-detecting member fields.
   *
   * @since 1.5
   */
  Visibility fieldVisibility() default Visibility.DEFAULT;
}
/**
* Public core annotations, most of which are used to configure how
* Data Mapping/Binding works. Annotations in this package can only
* have dependencies to non-annotation classes in Core package;
* annotations that have dependencies to Mapper classes are included
* in Mapper module (under <code>org.codehaus.jackson.map.annotate</code>).
* Also contains parameter types (mostly enums) needed by annotations.
*<p>
* In future (version 2.0?), this package will probably be split off
* as a separate jar/module, to allow use of annotations without
* including core module. This would be useful for third party value
* classes that themselves do not depend on Jackson, but may want to
* be annotated to be automatically and conveniently serializable by
* Jackson.
*/
package org.codehaus.jackson.annotate;
package org.codehaus.jackson.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
* Annotation that can be used to either suppress serialization of
* properties (during serialization), or ignore processing of
* JSON properties read (during deserialization).
*<p>
* Example:
*<pre>
* // to prevent specified fields from being serialized or deserialized
* // (i.e. not include in JSON output; or being set even if they were included)
* \@JsonIgnoreProperties({ "internalId", "secretKey" })
* // To ignore any unknown properties in JSON input without exception:
* \@JsonIgnoreProperties(ignoreUnknown=true)
*</pre>
*<p>
* Only applicable to classes, not for properties (getters, setters, fields).
*
* @since 1.4
```

```
*/
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonIgnoreProperties
{
  /**
   * Names of properties to ignore.
   */
  public String[] value() default { };

  /**
   * Property that defines whether it is ok to just ignore any
   * unrecognized properties during deserialization.
   * If true, all properties that are unrecognized -- that is,
   * there are no setters or creators that accept them -- are
   * ignored without warnings (although handlers for unknown
   * properties, if any, will still be called) without
   * exception.
   *<p>
   * Does not have any effect on serialization.
   */
  public boolean ignoreUnknown() default false;
}
/* Jackson JSON-processor.
*
* Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi
*
* Licensed under the License specified in file LICENSE, included with
* the source code and binary code bundles.
* You may not use this file except in compliance with the License.
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package org.codehaus.jackson;

import java.io.*;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.io.SerializedString;

/**
* Base class that defines public API for writing JSON content.
```

```
* Instances are created using factory methods of
* a {@link JsonFactory} instance.
*
* @author Tatu Saloranta
*/
public abstract class JsonGenerator
  implements Closeable, Versioned
{
  /**
   * Enumeration that defines all togglable features for generators.
   */
  public enum Feature {
    /**
     * Feature that determines whether generator will automatically
     * close underlying output target that is NOT owned by the
     * generator.
     * If disabled, calling application has to separately
     * close the underlying {@link OutputStream} and {@link Writer}
     * instances used to create the generator. If enabled, generator
     * will handle closing, as long as generator itself gets closed:
     * this happens when end-of-input is encountered, or generator
     * is closed by a call to {@link JsonGenerator#close}.
     *<p>
     * Feature is enabled by default.
     */
    AUTO_CLOSE_TARGET(true),

    /**
     * Feature that determines what happens when the generator is
     * closed while there are still unmatched
     * {@link JsonToken#START_ARRAY} or {@link JsonToken#START_OBJECT}
     * entries in output content. If enabled, such Array(s) and/or
     * Object(s) are automatically closed; if disabled, nothing
     * specific is done.
     *<p>
     * Feature is enabled by default.
     */
    AUTO_CLOSE_JSON_CONTENT(true),

    /**
     * Feature that determines whether JSON Object field names are
     * quoted using double-quotes, as specified by JSON specification
     * or not. Ability to disable quoting was added to support use
     * cases where they are not usually expected, which most commonly
     * occurs when used straight from Javascript.
     */
    QUOTE_FIELD_NAMES(true),
```

```
/**
 * Feature that determines whether "exceptional" (not real number)
 * float/double values are output as quoted strings.
 * The values checked are Double.Nan,
 * Double.POSITIVE_INFINITY and Double.NEGATIVE_INIFINTY (and
 * associated Float values).
 * If feature is disabled, these numbers are still output using
 * associated literal values, resulting in non-conformant
 * output.
 *<p>
 * Feature is enabled by default.
 */
QUOTE_NON_NUMERIC_NUMBERS(true),

/**
 * Feature that forces all Java numbers to be written as JSON strings.
 * Default state is 'false', meaning that Java numbers are to
 * be serialized using basic numeric serialization (as JSON
 * numbers, integral or floating point). If enabled, all such
 * numeric values are instead written out as JSON Strings.
 *<p>
 * One use case is to avoid problems with Javascript limitations:
 * since Javascript standard specifies that all number handling
 * should be done using 64-bit IEEE 754 floating point values,
 * result being that some 64-bit integer values can not be
 * accurately represent (as mantissa is only 51 bit wide).
 *<p>
 * Feature is disabled by default.
 *
 * @since 1.3
 */
WRITE_NUMBERS_AS_STRINGS(false),

/**
 * Feature that specifies that calls to {@link #flush} will cause
 * matching <code>flush()</code> to underlying {@link OutputStream}
 * or {@link Writer}; if disabled this will not be done.
 * Main reason to disable this feature is to prevent flushing at
 * generator level, if it is not possible to prevent method being
 * called by other code (like <code>ObjectMapper</code> or third
 * party libraries).
 *<p>
 * Feature is enabled by default.
 *
 * @since 1.7
 */
FLUSH_PASSED_TO_STREAM(true)
```

```
        ;

    final boolean _defaultState;

    final int _mask;

    /**
     * Method that calculates bit set (flags) of all features that
     * are enabled by default.
     */
    public static int collectDefaults()
    {
        int flags = 0;
        for (Feature f : values()) {
            if (f.enabledByDefault()) {
                flags |= f.getMask();
            }
        }
        return flags;
    }

    private Feature(boolean defaultState) {
        _defaultState = defaultState;
        _mask = (1 << ordinal());
    }

    public boolean enabledByDefault() { return _defaultState; }

    public int getMask() { return _mask; }
}

// // // Configuration:

/**
 * Object that handles pretty-printing (usually additional
 * white space to make results more human-readable) during
 * output. If null, no pretty-printing is done.
 */
protected PrettyPrinter _cfgPrettyPrinter;

protected JsonGenerator() {
}

/**
 * @since 1.6
 */
public Version version() {
    return Version.unknownVersion();
```

```
}

/*
/**********************************************************
/* Public API, configuration
/**********************************************************
 */

/**
 * Method for enabling specified parser features:
 * check {@link Feature} for list of available features.
 *
 * @return Generator itself (this), to allow chaining
 *
 * @since 1.2
 */
public abstract JsonGenerator enable(Feature f);

/**
 * Method for disabling specified  features
 * (check {@link Feature} for list of features)
 *
 * @return Generator itself (this), to allow chaining
 *
 * @since 1.2
 */
public abstract JsonGenerator disable(Feature f);

/**
 * Method for enabling or disabling specified feature:
 * check {@link Feature} for list of available features.
 *
 * @return Generator itself (this), to allow chaining
 *
 * @since 1.2
 */
public JsonGenerator configure(Feature f, boolean state)
{
   if (state) {
      enable(f);
   } else {
      disable(f);
   }
   return this;
}

/**
 * Method for checking whether given feature is enabled.
```

```
     * Check {@link Feature} for list of available features.
     *
     * @since 1.2
     */
    public abstract boolean isEnabled(Feature f);

    /**
     * Method that can be called to set or reset the object to
     * use for writing Java objects as JsonContent
     * (using method {@link #writeObject}).
     *
     * @return Generator itself (this), to allow chaining
     */
    public abstract JsonGenerator setCodec(ObjectCodec oc);

    /**
     * Method for accessing the object used for writing Java
     * object as Json content
     * (using method {@link #writeObject}).
     */
    public abstract ObjectCodec getCodec();

    // // // Older deprecated versions

    /** @deprecated Use {@link #enable} instead
     */
    @SuppressWarnings("dep-ann")
    public void enableFeature(Feature f) { enable(f); }

    /** @deprecated Use {@link #disable} instead
     */
    @SuppressWarnings("dep-ann")
    public void disableFeature(Feature f) { disable(f); }

    /** @deprecated Use {@link #configure} instead
     */
    @SuppressWarnings("dep-ann")
    public void setFeature(Feature f, boolean state) { configure(f, state); }

    /** @deprecated Use {@link #isEnabled} instead
     */
    @SuppressWarnings("dep-ann")
    public boolean isFeatureEnabled(Feature f) { return isEnabled(f); }

    /*
    /**********************************************************
    /* Configuring generator
    /**********************************************************
```

```
     */

    /**
     * Method for setting a custom pretty printer, which is usually
     * used to add indentation for improved human readability.
     * By default, generator does not do pretty printing.
     *<p>
     * To use the default pretty printer that comes with core
     * Jackson distribution, call {@link #useDefaultPrettyPrinter}
     * instead.
     *
     * @return Generator itself (this), to allow chaining
     */
    public JsonGenerator setPrettyPrinter(PrettyPrinter pp) {
        _cfgPrettyPrinter = pp;
        return this;
    }

    /**
     * Convenience method for enabling pretty-printing using
     * the default pretty printer
     * ({@link org.codehaus.jackson.util.DefaultPrettyPrinter}).
     *
     * @return Generator itself (this), to allow chaining
     */
    public abstract JsonGenerator useDefaultPrettyPrinter();

    /*
    /**********************************************************
    /* Public API, write methods, structural
    /**********************************************************
     */

    /**
     * Method for writing starting marker of a JSON Array value
     * (character '['; plus possible white space decoration
     * if pretty-printing is enabled).
     *<p>
     * Array values can be written in any context where values
     * are allowed: meaning everywhere except for when
     * a field name is expected.
     */
    public abstract void writeStartArray()
        throws IOException, JsonGenerationException;

    /**
     * Method for writing closing marker of a JSON Array value
     * (character ']'; plus possible white space decoration
```

```
     * if pretty-printing is enabled).
     *<p>
     * Marker can be written if the innermost structured type
     * is Array.
     */
    public abstract void writeEndArray()
        throws IOException, JsonGenerationException;

    /**
     * Method for writing starting marker of a JSON Object value
     * (character '{'; plus possible white space decoration
     * if pretty-printing is enabled).
     *<p>
     * Object values can be written in any context where values
     * are allowed: meaning everywhere except for when
     * a field name is expected.
     */
    public abstract void writeStartObject()
        throws IOException, JsonGenerationException;

    /**
     * Method for writing closing marker of a JSON Object value
     * (character '}'; plus possible white space decoration
     * if pretty-printing is enabled).
     *<p>
     * Marker can be written if the innermost structured type
     * is Object, and the last written event was either a
     * complete value, or START-OBJECT marker (see JSON specification
     * for more details).
     */
    public abstract void writeEndObject()
        throws IOException, JsonGenerationException;

    /**
     * Method for writing a field name (JSON String surrounded by
     * double quotes: syntactically identical to a JSON String value),
     * possibly decorated by white space if pretty-printing is enabled.
     *<p>
     * Field names can only be written in Object context (check out
     * JSON specification for details), when field name is expected
     * (field names alternate with values).
     */
    public abstract void writeFieldName(String name)
        throws IOException, JsonGenerationException;

    /**
     * Method similar to {@link #writeFieldName(String)}, main difference
     * being that it may perform better as some of processing (such as
```

```
     * quoting of certain characters, or encoding into external encoding
     * if supported by generator) can be done just once and reused for
     * later calls.
     *<p>
     * Default implementation simple uses unprocessed name container in
     * serialized String; implementations are strongly encouraged to make
     * use of more efficient methods argument object has.
     *
     * @since 1.6
     */
    public void writeFieldName(SerializedString name)
       throws IOException, JsonGenerationException
    {
       writeFieldName(name.getValue());
    }

    /**
     * Method similar to {@link #writeFieldName(String)}, main difference
     * being that it may perform better as some of processing (such as
     * quoting of certain characters, or encoding into external encoding
     * if supported by generator) can be done just once and reused for
     * later calls.
     *<p>
     * Default implementation simple uses unprocessed name container in
     * serialized String; implementations are strongly encouraged to make
     * use of more efficient methods argument object has.
     *
     * @since 1.7
     */
    public void writeFieldName(SerializableString name)
       throws IOException, JsonGenerationException
    {
       writeFieldName(name.getValue());
    }

    /*
    /**********************************************************
    /* Public API, write methods, text/String values
    /**********************************************************
     */

    /**
     * Method for outputting a String value. Depending on context
     * this means either array element, (object) field value or
     * a stand alone String; but in all cases, String will be
     * surrounded in double quotes, and contents will be properly
     * escaped as required by JSON specification.
     */
```

```
public abstract void writeString(String text)
   throws IOException, JsonGenerationException;

/**
 * Method for outputting a String value. Depending on context
 * this means either array element, (object) field value or
 * a stand alone String; but in all cases, String will be
 * surrounded in double quotes, and contents will be properly
 * escaped as required by JSON specification.
 */
public abstract void writeString(char[] text, int offset, int len)
   throws IOException, JsonGenerationException;

/**
 * Method similar to {@link #writeString(String)}, but that takes
 * {@link SerializableString} which can make this potentially
 * more efficient to call as generator may be able to reuse
 * quoted and/or encoded representation.
 *<p>
 * Default implementation just calls {@link #writeString(String)};
 * sub-classes should override it with more efficient implementation
 * if possible.
 *
 * @since 1.7
 */
public void writeString(SerializableString text)
   throws IOException, JsonGenerationException
{
   writeString(text.getValue());
}

/**
 * Method similar to {@link #writeString(String)} but that takes as
 * its input a UTF-8 encoded String that is to be output as-is, without additional
 * escaping (type of which depends on data format; backslashes for JSON).
 * However, quoting that data format requires (like double-quotes for JSON) will be added
 * around the value if and as necessary.
 *<p>
 * Note that some backends may choose not to support this method: for
 * example, if underlying destination is a {@link java.io.Writer}
 * using this method would require UTF-8 decoding.
 * If so, implementation may instead choose to throw a
 * {@link UnsupportedOperationException} due to ineffectiveness
 * of having to decode input.
 *
 * @since 1.7
 */
public abstract void writeRawUTF8String(byte[] text, int offset, int length)
```

```
    throws IOException, JsonGenerationException;

/**
 * Method similar to {@link #writeString(String)} but that takes as its input
 * a UTF-8 encoded String which has <b>not</b> been escaped using whatever
 * escaping scheme data format requires (for JSON that is backslash-escaping
 * for control characters and double-quotes; for other formats something else).
 * This means that textual JSON backends need to check if value needs
 * JSON escaping, but otherwise can just be copied as is to output.
 * Also, quoting that data format requires (like double-quotes for JSON) will be added
 * around the value if and as necessary.
 *<p>
 * Note that some backends may choose not to support this method: for
 * example, if underlying destination is a {@link java.io.Writer}
 * using this method would require UTF-8 decoding.
 * In this case
 * generator implementation may instead choose to throw a
 * {@link UnsupportedOperationException} due to ineffectiveness
 * of having to decode input.
 *
 * @since 1.7
 */
public abstract void writeUTF8String(byte[] text, int offset, int length)
    throws IOException, JsonGenerationException;

/*
/**********************************************************
/* Public API, write methods, binary/raw content
/**********************************************************
 */

/**
 * Method that will force generator to copy
 * input text verbatim with <b>no</b> modifications (including
 * that no escaping is done and no separators are added even
 * if context [array, object] would otherwise require such).
 * If such separators are desired, use
 * {@link #writeRawValue(String)} instead.
 *<p>
 * Note that not all generator implementations necessarily support
 * such by-pass methods: those that do not will throw
 * {@link UnsupportedOperationException}.
 */
public abstract void writeRaw(String text)
    throws IOException, JsonGenerationException;

/**
 * Method that will force generator to copy
```

```
     * input text verbatim with <b>no</b> modifications (including
     * that no escaping is done and no separators are added even
     * if context [array, object] would otherwise require such).
     * If such separators are desired, use
     * {@link #writeRawValue(String)} instead.
     *<p>
     * Note that not all generator implementations necessarily support
     * such by-pass methods: those that do not will throw
     * {@link UnsupportedOperationException}.
     */
    public abstract void writeRaw(String text, int offset, int len)
        throws IOException, JsonGenerationException;

    /**
     * Method that will force generator to copy
     * input text verbatim with <b>no</b> modifications (including
     * that no escaping is done and no separators are added even
     * if context [array, object] would otherwise require such).
     * If such separators are desired, use
     * {@link #writeRawValue(String)} instead.
     *<p>
     * Note that not all generator implementations necessarily support
     * such by-pass methods: those that do not will throw
     * {@link UnsupportedOperationException}.
     */
    public abstract void writeRaw(char[] text, int offset, int len)
        throws IOException, JsonGenerationException;

    /**
     * Method that will force generator to copy
     * input text verbatim with <b>no</b> modifications (including
     * that no escaping is done and no separators are added even
     * if context [array, object] would otherwise require such).
     * If such separators are desired, use
     * {@link #writeRawValue(String)} instead.
     *<p>
     * Note that not all generator implementations necessarily support
     * such by-pass methods: those that do not will throw
     * {@link UnsupportedOperationException}.
     */
    public abstract void writeRaw(char c)
        throws IOException, JsonGenerationException;

    /**
     * Method that will force generator to copy
     * input text verbatim without any modifications, but assuming
     * it must constitute a single legal JSON value (number, string,
     * boolean, null, Array or List). Assuming this, proper separators
```

```
     * are added if and as needed (comma or colon), and generator
     * state updated to reflect this.
     */
    public abstract void writeRawValue(String text)
       throws IOException, JsonGenerationException;

    public abstract void writeRawValue(String text, int offset, int len)
       throws IOException, JsonGenerationException;

    public abstract void writeRawValue(char[] text, int offset, int len)
       throws IOException, JsonGenerationException;

    /**
     * Method that will output given chunk of binary data as base64
     * encoded, as a complete String value (surrounded by double quotes).
     * This method defaults
     *<p>
     * Note: because Json Strings can not contain unescaped linefeeds,
     * if linefeeds are included (as per last argument), they must be
     * escaped. This adds overhead for decoding without improving
     * readability.
     * Alternatively if linefeeds are not included,
     * resulting String value may violate the requirement of base64
     * RFC which mandates line-length of 76 characters and use of
     * linefeeds. However, all {@link JsonParser} implementations
     * are required to accept such "long line base64"; as do
     * typical production-level base64 decoders.
     *
     * @param b64variant Base64 variant to use: defines details such as
     *   whether padding is used (and if so, using which character);
     *   what is the maximum line length before adding linefeed,
     *   and also the underlying alphabet to use.
     */
    public abstract void writeBinary(Base64Variant b64variant,
                                     byte[] data, int offset, int len)
       throws IOException, JsonGenerationException;

    /**
     * Similar to {@link #writeBinary(Base64Variant,byte[],int,int)},
     * but default to using the Jackson default Base64 variant
     * (which is {@link Base64Variants#MIME_NO_LINEFEEDS}).
     */
    public void writeBinary(byte[] data, int offset, int len)
       throws IOException, JsonGenerationException
    {
       writeBinary(Base64Variants.getDefaultVariant(), data, offset, len);
    }
```

```
/**
 * Similar to {@link #writeBinary(Base64Variant,byte[],int,int)},
 * but assumes default to using the Jackson default Base64 variant
 * (which is {@link Base64Variants#MIME_NO_LINEFEEDS}). Also
 * assumes that whole byte array is to be output.
 */
public void writeBinary(byte[] data)
   throws IOException, JsonGenerationException
{
   writeBinary(Base64Variants.getDefaultVariant(), data, 0, data.length);
}

/*
/**********************************************************
/* Public API, write methods, other value types
/**********************************************************
 */

/**
 * Method for outputting given value as Json number.
 * Can be called in any context where a value is expected
 * (Array value, Object field value, root-level value).
 * Additional white space may be added around the value
 * if pretty-printing is enabled.
 */
public abstract void writeNumber(int v)
   throws IOException, JsonGenerationException;

/**
 * Method for outputting given value as Json number.
 * Can be called in any context where a value is expected
 * (Array value, Object field value, root-level value).
 * Additional white space may be added around the value
 * if pretty-printing is enabled.
 */
public abstract void writeNumber(long v)
   throws IOException, JsonGenerationException;

/**
 * Method for outputting given value as Json number.
 * Can be called in any context where a value is expected
 * (Array value, Object field value, root-level value).
 * Additional white space may be added around the value
 * if pretty-printing is enabled.
 */
public abstract void writeNumber(BigInteger v)
   throws IOException, JsonGenerationException;
```

```
/**
 * Method for outputting indicate Json numeric value.
 * Can be called in any context where a value is expected
 * (Array value, Object field value, root-level value).
 * Additional white space may be added around the value
 * if pretty-printing is enabled.
 */
public abstract void writeNumber(double d)
   throws IOException, JsonGenerationException;

/**
 * Method for outputting indicate Json numeric value.
 * Can be called in any context where a value is expected
 * (Array value, Object field value, root-level value).
 * Additional white space may be added around the value
 * if pretty-printing is enabled.
 */
public abstract void writeNumber(float f)
   throws IOException, JsonGenerationException;

/**
 * Method for outputting indicate Json numeric value.
 * Can be called in any context where a value is expected
 * (Array value, Object field value, root-level value).
 * Additional white space may be added around the value
 * if pretty-printing is enabled.
 */
public abstract void writeNumber(BigDecimal dec)
   throws IOException, JsonGenerationException;

/**
 * Write method that can be used for custom numeric types that can
 * not be (easily?) converted to "standard" Java number types.
 * Because numbers are not surrounded by double quotes, regular
 * {@link #writeString} method can not be used; nor
 * {@link #writeRaw} because that does not properly handle
 * value separators needed in Array or Object contexts.
 *<p>
 * Note: because of lack of type safety, some generator
 * implementations may not be able to implement this
 * method. For example, if a binary json format is used,
 * it may require type information for encoding; similarly
 * for generator-wrappers around Java objects or Json nodes.
 * If implementation does not implement this method,
 * it needs to throw {@link UnsupportedOperationException}.
 */
public abstract void writeNumber(String encodedValue)
   throws IOException, JsonGenerationException,
```

```
        UnsupportedOperationException;

    /**
     * Method for outputting literal Json boolean value (one of
     * Strings 'true' and 'false').
     * Can be called in any context where a value is expected
     * (Array value, Object field value, root-level value).
     * Additional white space may be added around the value
     * if pretty-printing is enabled.
     */
    public abstract void writeBoolean(boolean state)
       throws IOException, JsonGenerationException;

    /**
     * Method for outputting literal Json null value.
     * Can be called in any context where a value is expected
     * (Array value, Object field value, root-level value).
     * Additional white space may be added around the value
     * if pretty-printing is enabled.
     */
    public abstract void writeNull()
       throws IOException, JsonGenerationException;

    /*
    /**********************************************************
    /* Public API, write methods, serializing Java objects
    /**********************************************************
     */

    /**
     * Method for writing given Java object (POJO) as Json.
     * Exactly how the object gets written depends on object
     * in question (ad on codec, its configuration); for most
     * beans it will result in Json object, but for others Json
     * array, or String or numeric value (and for nulls, Json
     * null literal.
     * <b>NOTE</b>: generator must have its <b>object codec</b>
     * set to non-null value; for generators created by a mapping
     * factory this is the case, for others not.
     */
    public abstract void writeObject(Object pojo)
       throws IOException, JsonProcessingException;

    /**
     * Method for writing given JSON tree (expressed as a tree
     * where given JsonNode is the root) using this generator.
     * This will generally just call
     * {@link #writeObject} with given node, but is added
```

```
     * for convenience and to make code more explicit in cases
     * where it deals specifically with trees.
     */
    public abstract void writeTree(JsonNode rootNode)
        throws IOException, JsonProcessingException;

    /*
    /**********************************************************
    /* Public API, convenience field write methods
    /**********************************************************
     */

    /**
     * Convenience method for outputting a field entry ("member")
     * that has a String value. Equivalent to:
     *<pre>
     *  writeFieldName(fieldName);
     *  writeString(value);
     *</pre>
     *<p>
     * Note: many performance-sensitive implementations override this method
     */
    public void writeStringField(String fieldName, String value)
        throws IOException, JsonGenerationException
    {
        writeFieldName(fieldName);
        writeString(value);
    }

    /**
     * Convenience method for outputting a field entry ("member")
     * that has a boolean value. Equivalent to:
     *<pre>
     *  writeFieldName(fieldName);
     *  writeBoolean(value);
     *</pre>
     */
    public final void writeBooleanField(String fieldName, boolean value)
        throws IOException, JsonGenerationException
    {
        writeFieldName(fieldName);
        writeBoolean(value);
    }

    /**
     * Convenience method for outputting a field entry ("member")
     * that has JSON literal value null. Equivalent to:
     *<pre>
```

```
 *  writeFieldName(fieldName);
 *  writeNull();
 *</pre>
 */
public final void writeNullField(String fieldName)
   throws IOException, JsonGenerationException
{
   writeFieldName(fieldName);
   writeNull();
}
/**
 * Convenience method for outputting a field entry ("member")
 * that has the specified numeric value. Equivalent to:
 *<pre>
 *  writeFieldName(fieldName);
 *  writeNumber(value);
 *</pre>
 */
public final void writeNumberField(String fieldName, int value)
   throws IOException, JsonGenerationException
{
   writeFieldName(fieldName);
   writeNumber(value);
}

/**
 * Convenience method for outputting a field entry ("member")
 * that has the specified numeric value. Equivalent to:
 *<pre>
 *  writeFieldName(fieldName);
 *  writeNumber(value);
 *</pre>
 */
public final void writeNumberField(String fieldName, long value)
   throws IOException, JsonGenerationException
{
   writeFieldName(fieldName);
   writeNumber(value);
}

/**
 * Convenience method for outputting a field entry ("member")
 * that has the specified numeric value. Equivalent to:
 *<pre>
 *  writeFieldName(fieldName);
 *  writeNumber(value);
 *</pre>
 */
```

```
public final void writeNumberField(String fieldName, double value)
   throws IOException, JsonGenerationException
{
   writeFieldName(fieldName);
   writeNumber(value);
}

/**
 * Convenience method for outputting a field entry ("member")
 * that has the specified numeric value. Equivalent to:
 *<pre>
 *  writeFieldName(fieldName);
 *  writeNumber(value);
 *</pre>
 */
public final void writeNumberField(String fieldName, float value)
   throws IOException, JsonGenerationException
{
   writeFieldName(fieldName);
   writeNumber(value);
}

/**
 * Convenience method for outputting a field entry ("member")
 * that has the specified numeric value.
 * Equivalent to:
 *<pre>
 *  writeFieldName(fieldName);
 *  writeNumber(value);
 *</pre>
 */
public final void writeNumberField(String fieldName, BigDecimal value)
   throws IOException, JsonGenerationException
{
   writeFieldName(fieldName);
   writeNumber(value);
}

/**
 * Convenience method for outputting a field entry ("member")
 * that contains specified data in base64-encoded form.
 * Equivalent to:
 *<pre>
 *  writeFieldName(fieldName);
 *  writeBinary(value);
 *</pre>
 */
public final void writeBinaryField(String fieldName, byte[] data)
```

```
    throws IOException, JsonGenerationException
{
    writeFieldName(fieldName);
    writeBinary(data);
}

/**
 * Convenience method for outputting a field entry ("member")
 * (that will contain a JSON Array value), and the START_ARRAY marker.
 * Equivalent to:
 *<pre>
 *  writeFieldName(fieldName);
 *  writeStartArray();
 *</pre>
 *<p>
 * Note: caller still has to take care to close the array
 * (by calling {#link #writeEndArray}) after writing all values
 * of the value Array.
 */
public final void writeArrayFieldStart(String fieldName)
    throws IOException, JsonGenerationException
{
    writeFieldName(fieldName);
    writeStartArray();
}

/**
 * Convenience method for outputting a field entry ("member")
 * (that will contain a JSON Object value), and the START_OBJECT marker.
 * Equivalent to:
 *<pre>
 *  writeFieldName(fieldName);
 *  writeStartObject();
 *</pre>
 *<p>
 * Note: caller still has to take care to close the Object
 * (by calling {#link #writeEndObject}) after writing all
 * entries of the value Object.
 */
public final void writeObjectFieldStart(String fieldName)
    throws IOException, JsonGenerationException
{
    writeFieldName(fieldName);
    writeStartObject();
}

/**
 * Convenience method for outputting a field entry ("member")
```

```
 * that has contents of specific Java object as its value.
 * Equivalent to:
 *<pre>
 *  writeFieldName(fieldName);
 *  writeObject(pojo);
 *</pre>
 */
public final void writeObjectField(String fieldName, Object pojo)
   throws IOException, JsonProcessingException
{
   writeFieldName(fieldName);
   writeObject(pojo);
}

/*
/**********************************************************
/* Public API, copy-through methods
/**********************************************************
 */

/**
 * Method for copying contents of the current event that
 * the given parser instance points to.
 * Note that the method <b>will not</b> copy any other events,
 * such as events contained within Json Array or Object structures.
 *<p>
 * Calling this method will not advance the given
 * parser, although it may cause parser to internally process
 * more data (if it lazy loads contents of value events, for example)
 */
public abstract void copyCurrentEvent(JsonParser jp)
   throws IOException, JsonProcessingException;

/**
 * Method for copying contents of the current event
 * <b>and following events that it encloses</b>
 * the given parser instance points to.
 *<p>
 * So what constitutes enclosing? Here is the list of
 * events that have associated enclosed events that will
 * get copied:
 *<ul>
 * <li>{@link JsonToken#START_OBJECT}:
 *   all events up to and including matching (closing)
 *   {@link JsonToken#END_OBJECT} will be copied
 *  </li>
 * <li>{@link JsonToken#START_ARRAY}
 *   all events up to and including matching (closing)
```

```
     *   {@link JsonToken#END_ARRAY} will be copied
     *  </li>
     * <li>{@link JsonToken#FIELD_NAME} the logical value (which
     *   can consist of a single scalar value; or a sequence of related
     *   events for structured types (Json Arrays, Objects)) will
     *   be copied along with the name itself. So essentially the
     *   whole <b>field entry</b> (name and value) will be copied.
     *  </li>
     *</ul>
     *<p>
     * After calling this method, parser will point to the
     * <b>last event</b> that was copied. This will either be
     * the event parser already pointed to (if there were no
     * enclosed events), or the last enclosed event copied.
     */
    public abstract void copyCurrentStructure(JsonParser jp)
       throws IOException, JsonProcessingException;

    /*
    /**********************************************************
    /* Public API, context access
    /**********************************************************
     */

    /**
     * @return Context object that can give information about logical
     *   position within generated json content.
     */
    public abstract JsonStreamContext getOutputContext();

    /*
    /**********************************************************
    /* Public API, buffer handling
    /**********************************************************
     */

    /**
     * Method called to flush any buffered content to the underlying
     * target (output stream, writer), and to flush the target itself
     * as well.
     */
    public abstract void flush() throws IOException;

    /**
     * Method that can be called to determine whether this generator
     * is closed or not. If it is closed, no more output can be done.
     */
    public abstract boolean isClosed();
```

```
  /*
  /**********************************************************
  /* Closeable implementation
  /**********************************************************
   */

  /**
   * Method called to close this generator, so that no more content
   * can be written.
   *<p>
   * Whether the underlying target (stream, writer) gets closed depends
   * on whether this generator either manages the target (i.e. is the
   * only one with access to the target -- case if caller passes a
   * reference to the resource such as File, but not stream); or
   * has feature {@link Feature#AUTO_CLOSE_TARGET} enabled.
   * If either of above is true, the target is also closed. Otherwise
   * (not managing, feature not enabled), target is not closed.
   */
  public abstract void close()
      throws IOException;
}
package org.codehaus.jackson;

/**
* Exception type for parsing problems, used when non-well-formed content
* (content that does not conform to JSON syntax as per specification)
* is encountered.
*/
public class JsonParseException
  extends JsonProcessingException
{
  final static long serialVersionUID = 123; // Stupid eclipse...

  public JsonParseException(String msg, JsonLocation loc)
  {
      super(msg, loc);
  }

  public JsonParseException(String msg, JsonLocation loc, Throwable root)
  {
      super(msg, loc, root);
  }
}
package org.codehaus.jackson.type;

import java.lang.reflect.Modifier;
```

```
/**
* Base class for type token classes used both to contain information
* and as keys for deserializers.
*<p>
* Instances can (only) be constructed by
* {@link org.codehaus.jackson.map.type.TypeFactory}.
*/
public abstract class JavaType
{
  /**
   * This is the nominal type-erased Class that would be close to the
   * type represented (but not exactly type, due to type erasure: type
   * instance may have more information on this).
   * May be an interface or abstract class, so instantiation
   * may not be possible.
   */
  protected final Class<?> _class;

  protected final int _hashCode;

  /**
   * Optional handler (codec) that can be attached to indicate
   * what to use for handling (serializing, deserializing) values of
   * this specific type.
   *<p>
   * Note: untyped (i.e. caller has to cast) because it is used for
   * different kinds of handlers, with unrelated types.
   *
   * @since 1.3
   */
  protected Object _valueHandler;

  /**
   * Optional handler that can be attached to indicate how to handle
   * additional type metadata associated with this type.
   *<p>
   * Note: untyped (i.e. caller has to cast) because it is used for
   * different kinds of handlers, with unrelated types.
   *
   * @since 1.5
   */
  protected Object _typeHandler;

  /*
  /**********************************************************
  /* Life-cycle
  /**********************************************************
   */
```

```
protected JavaType(Class<?> clz, int hash)
{
    _class = clz;
    String name = clz.getName();
    _hashCode = name.hashCode() + hash;
}

/**
 * "Copy method" that will construct a new instance that is identical to
 * this instance, except that it will have specified type handler assigned.
 *
 * @return Newly created type instance
 *
 * @since 1.7
 */
public abstract JavaType withTypeHandler(Object h);

/**
 * "Copy method" that will construct a new instance that is identical to
 * this instance, except that its content type will have specified
 * type handler assigned.
 *
 * @return Newly created type instance
 *
 * @since 1.7
 */
public abstract JavaType withContentTypeHandler(Object h);

/**
 * Method that can be called to do a "narrowing" conversions; that is,
 * to return a type with a raw class that is assignable to the raw
 * class of this type. If this is not possible, an
 * {@link IllegalArgumentException} is thrown.
 * If class is same as the current raw class, instance itself is
 * returned.
 */
public final JavaType narrowBy(Class<?> subclass)
{
    // First: if same raw class, just return this instance
    if (subclass == _class) {
        return this;
    }
    // Otherwise, ensure compatibility
    _assertSubclass(subclass, _class);
    JavaType result = _narrow(subclass);
    if (_valueHandler != null) {
        result.setValueHandler(_valueHandler);
```

```
    }
    if (_typeHandler != null) {
        result = result.withTypeHandler(_typeHandler);
    }
    return result;
}

/**
 * More efficient version of {@link #narrowBy}, called by
 * internal framework in cases where compatibility checks
 * are to be skipped.
 *
 * @since 1.5
 */
public final JavaType forcedNarrowBy(Class<?> subclass)
{
    if (subclass == _class) { // can still optimize for simple case
        return this;
    }
    JavaType result = _narrow(subclass);
    if (_valueHandler != null) {
        result.setValueHandler(_valueHandler);
    }
    if (_typeHandler != null) {
        result = result.withTypeHandler(_typeHandler);
    }
    return result;
}

/**
 * Method that can be called to do a "widening" conversions; that is,
 * to return a type with a raw class that could be assigned from this
 * type.
 * If such conversion is not possible, an
 * {@link IllegalArgumentException} is thrown.
 * If class is same as the current raw class, instance itself is
 * returned.
 */
public final JavaType widenBy(Class<?> superclass)
{
    // First: if same raw class, just return this instance
    if (superclass == _class) {
        return this;
    }
    // Otherwise, ensure compatibility
    _assertSubclass(_class, superclass);
    return _widen(superclass);
}
```

```
    protected abstract JavaType _narrow(Class<?> subclass);

    /**
     *<p>
     * Default implementation is just to call {@link #_narrow}, since
     * underlying type construction is usually identical
     */
    protected JavaType _widen(Class<?> superclass) {
        return _narrow(superclass);
    }

    public abstract JavaType narrowContentsBy(Class<?> contentClass);

    /**
     * Method for assigning handler to associate with this type; or
     * if null passed, to remove such assignment
     *
     * @since 1.3
     */
    public void setValueHandler(Object h) {
        // sanity check, should be assigned just once
        if (h != null && _valueHandler != null) {
            throw new IllegalStateException("Trying to reset value handler for type ["+toString()
              +"]; old handler of type "+_valueHandler.getClass().getName()+", new handler of type
"+h.getClass().getName());
        }
        _valueHandler = h;
    }

    /**
     * Method for assigning type handler to associate with this type; or
     * if null passed, to remove such assignment
     *
     * @since 1.5
     *
     * @deprecated Used {@link #withTypeHandler} instead -- this method is dangerous as
     *   it changes state, whereas all other functionality is stateless
     */
    @Deprecated
    public void setTypeHandler(Object h)
    {
        // sanity check, should be assigned just once
        /* 03-Nov-2010: NOTE - some care has to be taken to ensure that types are not reused
         *   between requests; one case I had to fix was that of passing root type by ObjectWriter
         *   and ObjectReader (must clone/copy types!)
         */
        if (h != null && _typeHandler != null) {
```

```
            throw new IllegalStateException("Trying to reset type handler for type ["+toString()
              +"]; old handler of type "+_typeHandler.getClass().getName()+", new handler of type
"+h.getClass().getName());
        }
        _typeHandler = h;
    }

    /*
    /**********************************************************
    /* Public API
    /**********************************************************
     */

    public final Class<?> getRawClass() { return _class; }

    /**
     * Method that can be used to check whether this type has
     * specified Class as its type erasure. Put another way, returns
     * true if instantiation of this Type is given (type-erased) Class.
     */
    public final boolean hasRawClass(Class<?> clz) {
        return _class == clz;
    }

    /**
     * @return True if type represented is a container type; this includes
     *    array, Map and Collection types.
     */
    public abstract boolean isContainerType();

    public boolean isAbstract() {
        return Modifier.isAbstract(_class.getModifiers());
    }

    /**
     * @since 1.3
     */
    public boolean isConcrete() {
        int mod = _class.getModifiers();
        if ((mod & (Modifier.INTERFACE | Modifier.ABSTRACT)) == 0) {
            return true;
        }
        /* 19-Feb-2010, tatus: Holy mackarel; primitive types
         *    have 'abstract' flag set...
         */
        if (_class.isPrimitive()) {
            return true;
        }
```

```
    return false;
}

public boolean isThrowable() {
    return Throwable.class.isAssignableFrom(_class);
}

public boolean isArrayType() { return false; }

public final boolean isEnumType() { return _class.isEnum(); }

public final boolean isInterface() { return _class.isInterface(); }

public final boolean isPrimitive() { return _class.isPrimitive(); }

public final boolean isFinal() { return Modifier.isFinal(_class.getModifiers()); }

/**
 * Method that can be used to find out if the type directly declares generic
 * parameters (for its direct super-class and/or super-interfaces).
 *
 * @since 1.6
 */
public boolean hasGenericTypes()
{
    return containedTypeCount() > 0;
}

/**
 * Method for accessing key type for this type, assuming type
 * has such a concept (only Map types do)
 */
public JavaType getKeyType() { return null; }

/**
 * Method for accessing content type of this type, if type has
 * such a thing: simple types do not, structured types do
 * (like arrays, Collections and Maps)
 */
public JavaType getContentType() { return null; }

/**
 * Method for checking how many contained types this type
 * has. Contained types are usually generic types, so that
 * generic Maps have 2 contained types.
 *
 * @since 1.5
 */
```

```
public int containedTypeCount() { return 0; }

/**
 * Method for accessing definitions of contained ("child")
 * types.
 *
 * @param index Index of contained type to return
 *
 * @return Contained type at index, or null if no such type
 *    exists (no exception thrown)
 *
 * @since 1.5
 */
public JavaType containedType(int index) { return null; }

/**
 * Method for accessing name of type variable in indicated
 * position. If no name is bound, will use placeholders (derived
 * from 0-based index); if no type variable or argument exists
 * with given index, null is returned.
 *
 * @param index Index of contained type to return
 *
 * @return Contained type at index, or null if no such type
 *    exists (no exception thrown)
 *
 * @since 1.5
 */
public String containedTypeName(int index) { return null; }

/**
 * Method for accessing value handler associated with this type, if any
 *
 * @since 1.3
 */
@SuppressWarnings("unchecked")
public <T> T getValueHandler() { return (T) _valueHandler; }

/**
 * Method for accessing type handler associated with this type, if any
 *
 * @since 1.5
 */
@SuppressWarnings("unchecked")
public <T> T getTypeHandler() { return (T) _typeHandler; }

/**
 * Method that can be used to serialize type into form from which
```

```
     * it can be fully deserialized from at a later point (using
     * <code>TypeFactory</code> from mapper package).
     * For simple types this is same as calling
     * {@link Class#getName}, but for structured types it may additionally
     * contain type information about contents.
     *
     * @since 1.5
     */
    public abstract String toCanonical();

    /*
    /**********************************************************
    /* Support for producing signatures (1.6+)
    /**********************************************************
     */

    /**
     * Method for accessing signature that contains generic
     * type information, in form compatible with JVM 1.5
     * as per JLS. It is a superset of {@link #getErasedSignature},
     * in that generic information can be automatically removed
     * if necessary (just remove outermost
     * angle brackets along with content inside)
     *
     * @since 1.6
     */
    public String getGenericSignature() {
        StringBuilder sb = new StringBuilder(40);
        getGenericSignature(sb);
        return sb.toString();
    }

    /**
     *
     * @param sb StringBuilder to append signature to
     *
     * @return StringBuilder that was passed in; returned to allow
     * call chaining
     *
     * @since 1.6
     */
    public abstract StringBuilder getGenericSignature(StringBuilder sb);

    /**
     * Method for accessing signature without generic
     * type information, in form compatible with all versions
     * of JVM, and specifically used for type descriptions
     * when generating byte code.
```

```
     *
     * @since 1.6
     */
    public String getErasedSignature() {
        StringBuilder sb = new StringBuilder(40);
        getErasedSignature(sb);
        return sb.toString();
    }

    /**
     * Method for accessing signature without generic
     * type information, in form compatible with all versions
     * of JVM, and specifically used for type descriptions
     * when generating byte code.
     *
     * @param sb StringBuilder to append signature to
     *
     * @return StringBuilder that was passed in; returned to allow
     * call chaining
     *
     * @since 1.6
     */
    public abstract StringBuilder getErasedSignature(StringBuilder sb);

    /*
    /**********************************************************
    /* Helper methods
    /**********************************************************
     */

    protected void _assertSubclass(Class<?> subclass, Class<?> superClass)
    {
        if (!_class.isAssignableFrom(subclass)) {
            throw new IllegalArgumentException("Class "+subclass.getName()+" is not assignable to
"+_class.getName());
        }
    }

    /*
    /**********************************************************
    /* Standard methods; let's make them abstract to force override
    /**********************************************************
     */

    @Override
    public abstract String toString();

    @Override
```

```
  public abstract boolean equals(Object o);

  @Override
  public final int hashCode() { return _hashCode; }
}
package org.codehaus.jackson.type;

import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;

/**
* This class is used to pass full generics type information, and
* avoid problems with type erasure (that basically removes most
* usable type references from runtime Class objects).
* It is based on ideas from
* <a href="http://gafter.blogspot.com/2006/12/super-type-tokens.html"
* >http://gafter.blogspot.com/2006/12/super-type-tokens.html</a>,
* Additional idea (from a suggestion made in comments of the article)
* is to require bogus implementation of <code>Comparable</code>
* (any such generic interface would do, as long as it forces a method
* with generic type to be implemented).
* to ensure that a Type argument is indeed given.
*<p>
* Usage is by sub-classing: here is one way to instantiate reference
* to generic type <code>List<Integer></code>:
*<pre>
*  TypeReference ref = new TypeReference<List<Integer>>() { };
*</pre>
* which can be passed to methods that accept TypeReference.
*/
public abstract class TypeReference<T>
  implements Comparable<TypeReference<T>>
{
  final Type _type;

  protected TypeReference()
  {
     Type superClass = getClass().getGenericSuperclass();
     if (superClass instanceof Class<?>) { // sanity check, should never happen
        throw new IllegalArgumentException("Internal error: TypeReference constructed without actual type
information");
     }
     /* 22-Dec-2008, tatu: Not sure if this case is safe -- I suspect
      *   it is possible to make it fail?
      *   But let's deal with specifc
      *   case when we know an actual use case, and thereby suitable
      *   work arounds for valid case(s) and/or error to throw
      *   on invalid one(s).
```

```
         */
        _type = ((ParameterizedType) superClass).getActualTypeArguments()[0];
    }

    public Type getType() { return _type; }

    /**
     * The only reason we define this method (and require implementation
     * of <code>Comparable</code>) is to prevent constructing a
     * reference without type information.
     */
    public int compareTo(TypeReference<T> o) {
        // just need an implementation, not a good one... hence:
        return 0;
    }
}
/**
* Contains classes needed for type introspection, mostly used by data binding
* functionality. Most of this functionality is needed to properly handled
* generic types, and to simplify and unify processing of things Jackson needs
* to determine how contained types (of {@link java.util.Collection} and
* {@link java.util.Map} classes) are to be handled.
*/
package org.codehaus.jackson.type;
/* Jackson JSON-processor.
*
* Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi
*
* Licensed under the License specified in file LICENSE, included with
* the source code and binary code bundles.
* You may not use this file except in compliance with the License.
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package org.codehaus.jackson;

/**
* Container for commonly used Base64 variants.
*
* @author Tatu Saloranta
*/
public final class Base64Variants
{
    final static String STD_BASE64_ALPHABET =
```

```
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

 /**
  * This variant is what most people would think of "the standard"
  * Base64 encoding.
  *<p>
  * See <a href="">wikipedia Base64 entry</a> for details.
  *<p>
  * Note that although this can be thought of as the standard variant,
  * it is <b>not</b> the default for Jackson: no-linefeeds alternative
  * is because of JSON requirement of escaping all linefeeds.
  */
 public final static Base64Variant MIME;
 static {
    MIME = new Base64Variant("MIME", STD_BASE64_ALPHABET, true, '=', 76);
 }

 /**
  * Slightly non-standard modification of {@link #MIME} which does not
  * use linefeeds (max line length set to infinite). Useful when linefeeds
  * wouldn't work well (possibly in attributes), or for minor space savings
  * (save 1 linefeed per 76 data chars, ie. ~1.4% savings).
  */
 public final static Base64Variant MIME_NO_LINEFEEDS;
 static {
    MIME_NO_LINEFEEDS = new Base64Variant(MIME, "MIME-NO-LINEFEEDS", Integer.MAX_VALUE);
 }

 /**
  * This variant is the one that predates {@link #MIME}: it is otherwise
  * identical, except that it mandates shorter line length.
  */
 public final static Base64Variant PEM = new Base64Variant(MIME, "PEM", true, '=', 64);

 /**
  * This non-standard variant is usually used when encoded data needs to be
  * passed via URLs (such as part of GET request). It differs from the
  * base {@link #MIME} variant in multiple ways.
  * First, no padding is used: this also means that it generally can not
  * be written in multiple separate but adjacent chunks (which would not
  * be the usual use case in any case). Also, no linefeeds are used (max
  * line length set to infinite). And finally, two characters (plus and
  * slash) that would need quoting in URLs are replaced with more
  * optimal alternatives (hyphen and underscore, respectively).
  */
 public final static Base64Variant MODIFIED_FOR_URL;
 static {
    StringBuffer sb = new StringBuffer(STD_BASE64_ALPHABET);
```

```
        // Replace plus with hyphen, slash with underscore (and no padding)
        sb.setCharAt(sb.indexOf("+"), '-');
        sb.setCharAt(sb.indexOf("/"), '_');
        /* And finally, let's not split lines either, wouldn't work too
         * well with URLs
         */
        MODIFIED_FOR_URL = new Base64Variant("MODIFIED-FOR-URL", sb.toString(), false,
Base64Variant.PADDING_CHAR_NONE, Integer.MAX_VALUE);
    }

    /**
     * Method used to get the default variant ("MIME_NO_LINEFEEDS") for cases
     * where caller does not explicitly specify the variant.
     * We will prefer no-linefeed version because linefeeds in JSON values
     * must be escaped, making linefeed-containing variants sub-optimal.
     */
    public static Base64Variant getDefaultVariant() {
        return MIME_NO_LINEFEEDS;
    }
}
package org.codehaus.jackson;

/**
* Exception type for exceptions during JSON writing, such as trying
* to output  content in wrong context (non-matching end-array or end-object,
* for example).
*/
public class JsonGenerationException
    extends JsonProcessingException
{
    final static long serialVersionUID = 123; // Stupid eclipse...

    public JsonGenerationException(Throwable rootCause)
    {
        super(rootCause);
    }

    public JsonGenerationException(String msg)
    {
        super(msg, (JsonLocation)null);
    }

    public JsonGenerationException(String msg, Throwable rootCause)
    {
        super(msg, (JsonLocation)null, rootCause);
    }
}
```

```
package org.codehaus.jackson;

/**
* Enumeration that defines legal encodings that can be used
* for JSON content, based on list of allowed encodings from
* <a href="http://www.ietf.org/rfc/rfc4627.txt">JSON specification</a>.
*<p>
* Note: if application want to explicitly disregard Encoding
* limitations (to read in JSON encoded using an encoding not
* listed as allowed), they can use {@link java.io.Reader} /
* {@link java.io.Writer} instances as input
* source (or output target).
*/
public enum JsonEncoding {
  UTF8("UTF-8", false), // N/A for big-endian, really
     UTF16_BE("UTF-16BE", true),
     UTF16_LE("UTF-16LE", false),
     UTF32_BE("UTF-32BE", true),
     UTF32_LE("UTF-32LE", false)
     ;

  final String mJavaName;

  final boolean mBigEndian;

  JsonEncoding(String javaName, boolean bigEndian)
  {
     mJavaName = javaName;
     mBigEndian = bigEndian;
  }

  /**
   * Method for accessing encoding name that JDK will support.
   *
   * @return Matching encoding name that JDK will support.
   */
  public String getJavaName() { return mJavaName; }

  /**
   * Whether encoding is big-endian (if encoding supports such
   * notion). If no such distinction is made (as is the case for
   * {@link #UTF8}), return value is undefined.
   *
   * @return True for big-endian encodings; false for little-endian
   *   (or if not applicable)
   */
  public boolean isBigEndian() { return mBigEndian; }
}
```

```
/* Jackson JSON-processor.
*
* Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi
*
* Licensed under the License specified in file LICENSE, included with
* the source code and binary code bundles.
* You may not use this file except in compliance with the License.
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package org.codehaus.jackson;

import java.util.Arrays;

/**
* Abstract base class used to define specific details of which
* variant of Base64 encoding/decoding is to be used. Although there is
* somewhat standard basic version (so-called "MIME Base64"), other variants
* exists, see <a href="http://en.wikipedia.org/wiki/Base64">Base64 Wikipedia entry</a> for details.
*
* @author Tatu Saloranta
*
* @since 0.9.3
*/
public final class Base64Variant
{
  /**
   * Placeholder used by "no padding" variant, to be used when a character
   * value is needed.
   */
  final static char PADDING_CHAR_NONE = '\0';

  /**
   * Marker used to denote ascii characters that do not correspond
   * to a 6-bit value (in this variant), and is not used as a padding
   * character.
   */
  public final static int BASE64_VALUE_INVALID = -1;

  /**
   * Marker used to denote ascii character (in decoding table) that
   * is the padding character using this variant (if any).
   */
  public final static int BASE64_VALUE_PADDING = -2;
```

```
/*
/**********************************************************
/* Encoding/decoding tables
/**********************************************************
 */

/**
 * Decoding table used for base 64 decoding.
 */
private final int[] _asciiToBase64 = new int[128];

/**
 * Encoding table used for base 64 decoding when output is done
 * as characters.
 */
private final char[] _base64ToAsciiC = new char[64];

/**
 * Alternative encoding table used for base 64 decoding when output is done
 * as ascii bytes.
 */
private final byte[] _base64ToAsciiB = new byte[64];

/*
/**********************************************************
/* Other configuration
/**********************************************************
 */

/**
 * Symbolic name of variant; used for diagnostics/debugging.
 */
final String _name;

/**
 * Whether this variant uses padding or not.
 */
final boolean _usesPadding;

/**
 * Characted used for padding, if any ({@link #PADDING_CHAR_NONE} if not).
 */
final char _paddingChar;

/**
 * Maximum number of encoded base64 characters to output during encoding
 * before adding a linefeed, if line length is to be limited
```

```
     * ({@link java.lang.Integer#MAX_VALUE} if not limited).
     *<p>
     * Note: for some output modes (when writing attributes) linefeeds may
     * need to be avoided, and this value ignored.
     */
    final int _maxLineLength;

    /*
    /**********************************************************
    /* Life-cycle
    /**********************************************************
     */

    public Base64Variant(String name, String base64Alphabet, boolean usesPadding, char paddingChar, int
maxLineLength)
    {
        _name = name;
        _usesPadding = usesPadding;
        _paddingChar = paddingChar;
        _maxLineLength = maxLineLength;

        // Ok and then we need to create codec tables.

        // First the main encoding table:
        int alphaLen = base64Alphabet.length();
        if (alphaLen != 64) {
            throw new IllegalArgumentException("Base64Alphabet length must be exactly 64 (was "+alphaLen+")");
        }

        // And then secondary encoding table and decoding table:
        base64Alphabet.getChars(0, alphaLen, _base64ToAsciiC, 0);
        Arrays.fill(_asciiToBase64, BASE64_VALUE_INVALID);
        for (int i = 0; i < alphaLen; ++i) {
            char alpha = _base64ToAsciiC[i];
            _base64ToAsciiB[i] = (byte) alpha;
            _asciiToBase64[alpha] = i;
        }

        // Plus if we use padding, add that in too
        if (usesPadding) {
            _asciiToBase64[(int) paddingChar] = BASE64_VALUE_PADDING;
        }
    }

    /**
     * "Copy constructor" that can be used when the base alphabet is identical
     * to one used by another variant except for the maximum line length
     * (and obviously, name).
```

```
     */
    public Base64Variant(Base64Variant base, String name, int maxLineLength)
    {
        this(base, name, base._usesPadding, base._paddingChar, maxLineLength);
    }

    /**
     * "Copy constructor" that can be used when the base alphabet is identical
     * to one used by another variant, but other details (padding, maximum
     * line length) differ
     */
    public Base64Variant(Base64Variant base, String name, boolean usesPadding, char paddingChar, int
maxLineLength)
    {
        _name = name;
        byte[] srcB = base._base64ToAsciiB;
        System.arraycopy(srcB, 0, this._base64ToAsciiB, 0, srcB.length);
        char[] srcC = base._base64ToAsciiC;
        System.arraycopy(srcC, 0, this._base64ToAsciiC, 0, srcC.length);
        int[] srcV = base._asciiToBase64;
        System.arraycopy(srcV, 0, this._asciiToBase64, 0, srcV.length);

        _usesPadding = usesPadding;
        _paddingChar = paddingChar;
        _maxLineLength = maxLineLength;
    }

    /*
    /**********************************************************
    /* Public accessors
    /**********************************************************
     */

    public String getName() { return _name; }

    public boolean usesPadding() { return _usesPadding; }
    public boolean usesPaddingChar(char c) { return c == _paddingChar; }
    public boolean usesPaddingChar(int ch) { return ch == (int) _paddingChar; }
    public char getPaddingChar() { return _paddingChar; }
    public byte getPaddingByte() { return (byte)_paddingChar; }

    public int getMaxLineLength() { return _maxLineLength; }

    /*
    /**********************************************************
    /* Decoding support
    /**********************************************************
     */
```

```
/**
 * @return 6-bit decoded value, if valid character;
 */
public int decodeBase64Char(char c)
{
   int ch = (int) c;
   return (ch <= 127) ? _asciiToBase64[ch] : BASE64_VALUE_INVALID;
}

public int decodeBase64Char(int ch)
{
   return (ch <= 127) ? _asciiToBase64[ch] : BASE64_VALUE_INVALID;
}

public int decodeBase64Byte(byte b)
{
   int ch = (int) b;
   return (ch <= 127) ? _asciiToBase64[ch] : BASE64_VALUE_INVALID;
}

/*
/**********************************************************
/* Encoding support
/**********************************************************
 */

public char encodeBase64BitsAsChar(int value)
{
   /* Let's assume caller has done necessary checks; this
    * method must be fast and inlinable
    */
   return _base64ToAsciiC[value];
}

/**
 * Method that encodes given right-aligned (LSB) 24-bit value
 * into 4 base64 characters, stored in given result buffer.
 */
public int encodeBase64Chunk(int b24, char[] buffer, int ptr)
{
   buffer[ptr++] = _base64ToAsciiC[(b24 >> 18) & 0x3F];
   buffer[ptr++] = _base64ToAsciiC[(b24 >> 12) & 0x3F];
   buffer[ptr++] = _base64ToAsciiC[(b24 >> 6) & 0x3F];
   buffer[ptr++] = _base64ToAsciiC[b24 & 0x3F];
   return ptr;
}
```

```
public void encodeBase64Chunk(StringBuilder sb, int b24)
{
   sb.append(_base64ToAsciiC[(b24 >> 18) & 0x3F]);
   sb.append(_base64ToAsciiC[(b24 >> 12) & 0x3F]);
   sb.append(_base64ToAsciiC[(b24 >> 6) & 0x3F]);
   sb.append(_base64ToAsciiC[b24 & 0x3F]);
}

/**
 * Method that outputs partial chunk (which only encodes one
 * or two bytes of data). Data given is still aligned same as if
 * it as full data; that is, missing data is at the "right end"
 * (LSB) of int.
 *
 * @param outputBytes Number of encoded bytes included (either 1 or 2)
 */
public int encodeBase64Partial(int bits, int outputBytes, char[] buffer, int outPtr)
{
   buffer[outPtr++] = _base64ToAsciiC[(bits >> 18) & 0x3F];
   buffer[outPtr++] = _base64ToAsciiC[(bits >> 12) & 0x3F];
   if (_usesPadding) {
      buffer[outPtr++] = (outputBytes == 2) ?
         _base64ToAsciiC[(bits >> 6) & 0x3F] : _paddingChar;
      buffer[outPtr++] = _paddingChar;
   } else {
      if (outputBytes == 2) {
         buffer[outPtr++] = _base64ToAsciiC[(bits >> 6) & 0x3F];
      }
   }
   return outPtr;
}

public void encodeBase64Partial(StringBuilder sb, int bits, int outputBytes)
{
   sb.append(_base64ToAsciiC[(bits >> 18) & 0x3F]);
   sb.append(_base64ToAsciiC[(bits >> 12) & 0x3F]);
   if (_usesPadding) {
      sb.append((outputBytes == 2) ?
            _base64ToAsciiC[(bits >> 6) & 0x3F] : _paddingChar);
      sb.append(_paddingChar);
   } else {
      if (outputBytes == 2) {
         sb.append(_base64ToAsciiC[(bits >> 6) & 0x3F]);
      }
   }
}

public byte encodeBase64BitsAsByte(int value)
```

```
{
    // As with above, assuming it is 6-bit value
    return _base64ToAsciiB[value];
}

/**
 * Method that encodes given right-aligned (LSB) 24-bit value
 * into 4 base64 bytes (ascii), stored in given result buffer.
 */
public int encodeBase64Chunk(int b24, byte[] buffer, int ptr)
{
    buffer[ptr++] = _base64ToAsciiB[(b24 >> 18) & 0x3F];
    buffer[ptr++] = _base64ToAsciiB[(b24 >> 12) & 0x3F];
    buffer[ptr++] = _base64ToAsciiB[(b24 >> 6) & 0x3F];
    buffer[ptr++] = _base64ToAsciiB[b24 & 0x3F];
    return ptr;
}

/**
 * Method that outputs partial chunk (which only encodes one
 * or two bytes of data). Data given is still aligned same as if
 * it as full data; that is, missing data is at the "right end"
 * (LSB) of int.
 *
 * @param outputBytes Number of encoded bytes included (either 1 or 2)
 */
public int encodeBase64Partial(int bits, int outputBytes, byte[] buffer, int outPtr)
{
    buffer[outPtr++] = _base64ToAsciiB[(bits >> 18) & 0x3F];
    buffer[outPtr++] = _base64ToAsciiB[(bits >> 12) & 0x3F];
    if (_usesPadding) {
        byte pb = (byte) _paddingChar;
        buffer[outPtr++] = (outputBytes == 2) ?
            _base64ToAsciiB[(bits >> 6) & 0x3F] : pb;
        buffer[outPtr++] = pb;
    } else {
        if (outputBytes == 2) {
            buffer[outPtr++] = _base64ToAsciiB[(bits >> 6) & 0x3F];
        }
    }
    return outPtr;
}

/**
 * Convenience method for converting given byte array as base64 encoded
 * String using this variant's settings.
 * Resulting value is "raw", that is, not enclosed in double-quotes.
 *
```

```
     * @param input Byte array to encode
     *
     * @since 1.6
     */
    public String encode(byte[] input)
    {
        return encode(input, false);
    }

    /**
     * Convenience method for converting given byte array as base64 encoded
     * String using this variant's settings, optionally enclosed in
     * double-quotes.
     *
     * @param input Byte array to encode
     * @param addQuotes Whether to surround resulting value in double quotes
     *    or not
     *
     * @since 1.6
     */
    public String encode(byte[] input, boolean addQuotes)
    {
        int inputEnd = input.length;
        StringBuilder sb;
        {
            // let's approximate... 33% overhead, ~= 3/8 (0.375)
            int outputLen = inputEnd + (inputEnd >> 2) + (inputEnd >> 3);
            sb = new StringBuilder(outputLen);
        }
        if (addQuotes) {
            sb.append('"');
        }

        int chunksBeforeLF = getMaxLineLength() >> 2;

        // Ok, first we loop through all full triplets of data:
        int inputPtr = 0;
        int safeInputEnd = inputEnd-3; // to get only full triplets

        while (inputPtr <= safeInputEnd) {
            // First, mash 3 bytes into lsb of 32-bit int
            int b24 = ((int) input[inputPtr++]) << 8;
            b24 |= ((int) input[inputPtr++]) & 0xFF;
            b24 = (b24 << 8) | (((int) input[inputPtr++]) & 0xFF);
            encodeBase64Chunk(sb, b24);
            if (--chunksBeforeLF <= 0) {
                // note: must quote in JSON value, so not really useful...
                sb.append('\\');
```

```
            sb.append('n');
            chunksBeforeLF = getMaxLineLength() >> 2;
          }
        }

        // And then we may have 1 or 2 leftover bytes to encode
        int inputLeft = inputEnd - inputPtr; // 0, 1 or 2
        if (inputLeft > 0) { // yes, but do we have room for output?
          int b24 = ((int) input[inputPtr++]) << 16;
          if (inputLeft == 2) {
            b24 |= (((int) input[inputPtr++]) & 0xFF) << 8;
          }
          encodeBase64Partial(sb, b24, inputLeft);
        }

        if (addQuotes) {
          sb.append('"');
        }
        return sb.toString();
      }

      /*
      /**********************************************************
      /* other methods
      /**********************************************************
       */

      @Override
      public String toString() { return _name; }
    }
    package org.codehaus.jackson.impl;

    import java.math.BigDecimal;
    import java.math.BigInteger;

    import java.io.*;

    import org.codehaus.jackson.*;
    import org.codehaus.jackson.io.IOContext;
    import org.codehaus.jackson.io.NumberInput;

    /**
    * Another intermediate base class used by all Jackson {@link JsonParser}
    * implementations. Contains shared functionality for dealing with
    * number parsing aspects, independent of input source decoding.
    *
    * @author Tatu Saloranta
    */
```

```
public abstract class JsonNumericParserBase
  extends JsonParserBase
{
  /* Additionally we need to be able to distinguish between
   * various numeric representations, since we try to use
   * the fastest one that works for given textual representation.
   */

  final protected static int NR_UNKNOWN = 0;

  // First, integer types

  final protected static int NR_INT = 0x0001;
  final protected static int NR_LONG = 0x0002;
  final protected static int NR_BIGINT = 0x0004;

  // And then floating point types

  final protected static int NR_DOUBLE = 0x008;
  final protected static int NR_BIGDECIMAL = 0x0010;

  // Also, we need some numeric constants

  final static BigDecimal BD_MIN_LONG = new BigDecimal(Long.MIN_VALUE);
  final static BigDecimal BD_MAX_LONG = new BigDecimal(Long.MAX_VALUE);

  final static BigDecimal BD_MIN_INT = new BigDecimal(Long.MIN_VALUE);
  final static BigDecimal BD_MAX_INT = new BigDecimal(Long.MAX_VALUE);

  final static long MIN_INT_L = (long) Integer.MIN_VALUE;
  final static long MAX_INT_L = (long) Integer.MAX_VALUE;

  // These are not very accurate, but have to do... (for bounds checks)

  final static double MIN_LONG_D = (double) Long.MIN_VALUE;
  final static double MAX_LONG_D = (double) Long.MAX_VALUE;

  final static double MIN_INT_D = (double) Integer.MIN_VALUE;
  final static double MAX_INT_D = (double) Integer.MAX_VALUE;


  // Digits, numeric
  final protected static int INT_0 = '0';
  final protected static int INT_1 = '1';
  final protected static int INT_2 = '2';
  final protected static int INT_3 = '3';
  final protected static int INT_4 = '4';
  final protected static int INT_5 = '5';
```

```
final protected static int INT_6 = '6';
final protected static int INT_7 = '7';
final protected static int INT_8 = '8';
final protected static int INT_9 = '9';

final protected static int INT_MINUS = '-';
final protected static int INT_PLUS = '+';
final protected static int INT_DECIMAL_POINT = '.';

final protected static int INT_e = 'e';
final protected static int INT_E = 'E';

final protected static char CHAR_NULL = '\0';

/*
/**********************************************************
/* Numeric value holders: multiple fields used for
/* for efficiency
/**********************************************************
 */

/**
 * Bitfield that indicates which numeric representations
 * have been calculated for the current type
 */
protected int _numTypesValid = NR_UNKNOWN;

// First primitives

protected int _numberInt;

protected long _numberLong;

protected double _numberDouble;

// And then object types

protected BigInteger _numberBigInt;

protected BigDecimal _numberBigDecimal;

// And then other information about value itself

/**
 * Flag that indicates whether numeric value has a negative
 * value. That is, whether its textual representation starts
 * with minus character.
 */
```

```
protected boolean _numberNegative;

/**
 * Length of integer part of the number, in characters
 */
protected int _intLength;

/**
 * Length of the fractional part (not including decimal
 * point or exponent), in characters.
 * Not used for  pure integer values.
 */
protected int _fractLength;

/**
 * Length of the exponent part of the number, if any, not
 * including 'e' marker or sign, just digits.
 * Not used for  pure integer values.
 */
protected int _expLength;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

protected JsonNumericParserBase(IOContext ctxt, int features)
{
   super(ctxt, features);
}

protected final JsonToken reset(boolean negative, int intLen, int fractLen, int expLen)
{
   if (fractLen < 1 && expLen < 1) { // integer
      return resetInt(negative, intLen);
   }
   return resetFloat(negative, intLen, fractLen, expLen);
}

protected final JsonToken resetInt(boolean negative, int intLen)
{
   _numberNegative = negative;
   _intLength = intLen;
   _fractLength = 0;
   _expLength = 0;
   _numTypesValid = NR_UNKNOWN; // to force parsing
   return JsonToken.VALUE_NUMBER_INT;
```

```
}

protected final JsonToken resetFloat(boolean negative, int intLen, int fractLen, int expLen)
{
   _numberNegative = negative;
   _intLength = intLen;
   _fractLength = fractLen;
   _expLength = expLen;
   _numTypesValid = NR_UNKNOWN; // to force parsing
   return JsonToken.VALUE_NUMBER_FLOAT;
}

/*
/**********************************************************
/* Numeric accessors of public API
/**********************************************************
 */

@Override
public Number getNumberValue() throws IOException, JsonParseException
{
   if (_numTypesValid == NR_UNKNOWN) {
      _parseNumericValue(NR_UNKNOWN); // will also check event type
   }
   // Separate types for int types
   if (_currToken == JsonToken.VALUE_NUMBER_INT) {
      if ((_numTypesValid & NR_INT) != 0) {
         return Integer.valueOf(_numberInt);
      }
      if ((_numTypesValid & NR_LONG) != 0) {
         return Long.valueOf(_numberLong);
      }
      if ((_numTypesValid & NR_BIGINT) != 0) {
         return _numberBigInt;
      }
      // Shouldn't get this far but if we do
      return _numberBigDecimal;
   }

   /* And then floating point types. But here optimal type
    * needs to be big decimal, to avoid losing any data?
    */
   if ((_numTypesValid & NR_BIGDECIMAL) != 0) {
      return _numberBigDecimal;
   }
   if ((_numTypesValid & NR_DOUBLE) == 0) { // sanity check
      _throwInternal();
   }
```

```
    return Double.valueOf(_numberDouble);
}

@Override
public NumberType getNumberType() throws IOException, JsonParseException
{
    if (_numTypesValid == NR_UNKNOWN) {
        _parseNumericValue(NR_UNKNOWN); // will also check event type
    }
    if (_currToken == JsonToken.VALUE_NUMBER_INT) {
        if ((_numTypesValid & NR_INT) != 0) {
            return NumberType.INT;
        }
        if ((_numTypesValid & NR_LONG) != 0) {
            return NumberType.LONG;
        }
        return NumberType.BIG_INTEGER;
    }

    /* And then floating point types. Here optimal type
     * needs to be big decimal, to avoid losing any data?
     * However... using BD is slow, so let's allow returning
     * double as type if no explicit call has been made to access
     * data as BD?
     */
    if ((_numTypesValid & NR_BIGDECIMAL) != 0) {
        return NumberType.BIG_DECIMAL;
    }
    return NumberType.DOUBLE;
}

@Override
public int getIntValue() throws IOException, JsonParseException
{
    if ((_numTypesValid & NR_INT) == 0) {
        if (_numTypesValid == NR_UNKNOWN) { // not parsed at all
            _parseNumericValue(NR_INT); // will also check event type
        }
        if ((_numTypesValid & NR_INT) == 0) { // wasn't an int natively?
            convertNumberToInt(); // let's make it so, if possible
        }
    }
    return _numberInt;
}

@Override
public long getLongValue() throws IOException, JsonParseException
{
```

```
    if ((_numTypesValid & NR_LONG) == 0) {
        if (_numTypesValid == NR_UNKNOWN) {
            _parseNumericValue(NR_LONG);
        }
        if ((_numTypesValid & NR_LONG) == 0) {
            convertNumberToLong();
        }
    }
    return _numberLong;
}

@Override
public BigInteger getBigIntegerValue() throws IOException, JsonParseException
{
    if ((_numTypesValid & NR_BIGINT) == 0) {
        if (_numTypesValid == NR_UNKNOWN) {
            _parseNumericValue(NR_BIGINT);
        }
        if ((_numTypesValid & NR_BIGINT) == 0) {
            convertNumberToBigInteger();
        }
    }
    return _numberBigInt;
}

@Override
public float getFloatValue() throws IOException, JsonParseException
{
    double value = getDoubleValue();
    /* 22-Jan-2009, tatu: Bounds/range checks would be tricky
     *   here, so let's not bother even trying...
     */
    /*
    if (value < -Float.MAX_VALUE || value > MAX_FLOAT_D) {
        _reportError("Numeric value ("+getText()+") out of range of Java float");
    }
    */
    return (float) value;
}

@Override
public double getDoubleValue() throws IOException, JsonParseException
{
    if ((_numTypesValid & NR_DOUBLE) == 0) {
        if (_numTypesValid == NR_UNKNOWN) {
            _parseNumericValue(NR_DOUBLE);
        }
        if ((_numTypesValid & NR_DOUBLE) == 0) {
```

```
            convertNumberToDouble();
        }
    }
    return _numberDouble;
}

@Override
public BigDecimal getDecimalValue() throws IOException, JsonParseException
{
    if ((_numTypesValid & NR_BIGDECIMAL) == 0) {
        if (_numTypesValid == NR_UNKNOWN) {
            _parseNumericValue(NR_BIGDECIMAL);
        }
        if ((_numTypesValid & NR_BIGDECIMAL) == 0) {
            convertNumberToBigDecimal();
        }
    }
    return _numberBigDecimal;
}



/*
/**********************************************************
/* Conversion from textual to numeric representation
/**********************************************************
 */

/**
 * Method that will parse actual numeric value out of a syntactically
 * valid number value. Type it will parse into depends on whether
 * it is a floating point number, as well as its magnitude: smallest
 * legal type (of ones available) is used for efficiency.
 *
 * @param expType Numeric type that we will immediately need, if any;
 *   mostly necessary to optimize handling of floating point numbers
 */
protected void _parseNumericValue(int expType)
    throws IOException, JsonParseException
{
    // Int or float?
    if (_currToken == JsonToken.VALUE_NUMBER_INT) {
        char[] buf = _textBuffer.getTextBuffer();
        int offset = _textBuffer.getTextOffset();
        int len = _intLength;
        if (_numberNegative) {
            ++offset;
        }
        if (len <= 9) { // definitely fits in int
```

```
            int i = NumberInput.parseInt(buf, offset, len);
            _numberInt = _numberNegative ? -i : i;
            _numTypesValid = NR_INT;
            return;
        }
        if (len <= 18) { // definitely fits AND is easy to parse using 2 int parse calls
            long l = NumberInput.parseLong(buf, offset, len);
            if (_numberNegative) {
                l = -l;
            }
            // [JACKSON-230] Could still fit in int, need to check
            if (len == 10) {
                if (_numberNegative) {
                    if (l >= MIN_INT_L) {
                        _numberInt = (int) l;
                        _numTypesValid = NR_INT;
                        return;
                    }
                } else {
                    if (l <= MAX_INT_L) {
                        _numberInt = (int) l;
                        _numTypesValid = NR_INT;
                        return;
                    }
                }
            }
            _numberLong = l;
            _numTypesValid = NR_LONG;
            return;
        }
        _parseSlowIntValue(expType, buf, offset, len);
        return;
    }
    if (_currToken == JsonToken.VALUE_NUMBER_FLOAT) {
        _parseSlowFloatValue(expType);
        return;
    }
    _reportError("Current token ("+_currToken+") not numeric, can not use numeric value accessors");
}

private final void _parseSlowFloatValue(int expType)
    throws IOException, JsonParseException
{
    /* Nope: floating point. Here we need to be careful to get
     * optimal parsing strategy: choice is between accurate but
     * slow (BigDecimal) and lossy but fast (Double). For now
     * let's only use BD when explicitly requested -- it can
     * still be constructed correctly at any point since we do
```

```
     * retain textual representation
     */
    try {
        if (expType == NR_BIGDECIMAL) {
            _numberBigDecimal = _textBuffer.contentsAsDecimal();
            _numTypesValid = NR_BIGDECIMAL;
        } else {
            // Otherwise double has to do
            _numberDouble = _textBuffer.contentsAsDouble();
            _numTypesValid = NR_DOUBLE;
        }
    } catch (NumberFormatException nex) {
        // Can this ever occur? Due to overflow, maybe?
        _wrapError("Malformed numeric value '"+_textBuffer.contentsAsString()+"'", nex);
    }
}

private final void _parseSlowIntValue(int expType, char[] buf, int offset, int len)
    throws IOException, JsonParseException
{
    String numStr = _textBuffer.contentsAsString();
    try {
        // [JACKSON-230] Some long cases still...
        if (NumberInput.inLongRange(buf, offset, len, _numberNegative)) {
            // Probably faster to construct a String, call parse, than to use BigInteger
            _numberLong = Long.parseLong(numStr);
            _numTypesValid = NR_LONG;
        } else {
            // nope, need the heavy guns... (rare case)
            _numberBigInt = new BigInteger(numStr);
            _numTypesValid = NR_BIGINT;
        }
    } catch (NumberFormatException nex) {
        // Can this ever occur? Due to overflow, maybe?
        _wrapError("Malformed numeric value '"+numStr+"'", nex);
    }
}

/*
/**********************************************************
/* Conversions
/**********************************************************
 */

protected void convertNumberToInt()
    throws IOException, JsonParseException
{
    // First, converting from long ought to be easy
```

```
    if ((_numTypesValid & NR_LONG) != 0) {
        // Let's verify it's lossless conversion by simple roundtrip
        int result = (int) _numberLong;
        if (((long) result) != _numberLong) {
            _reportError("Numeric value ("+getText()+") out of range of int");
        }
        _numberInt = result;
    } else if ((_numTypesValid & NR_BIGINT) != 0) {
        // !!! Should check for range...
        _numberInt = _numberBigInt.intValue();
    } else if ((_numTypesValid & NR_DOUBLE) != 0) {
        // Need to check boundaries
        if (_numberDouble < MIN_INT_D || _numberDouble > MAX_INT_D) {
            reportOverflowInt();
        }
        _numberInt = (int) _numberDouble;
    } else if ((_numTypesValid & NR_BIGDECIMAL) != 0) {
        if (BD_MIN_INT.compareTo(_numberBigDecimal) > 0
            || BD_MAX_INT.compareTo(_numberBigDecimal) < 0) {
            reportOverflowInt();
        }
        _numberInt = _numberBigDecimal.intValue();
    } else {
        _throwInternal(); // should never get here
    }

    _numTypesValid |= NR_INT;
}

protected void convertNumberToLong()
    throws IOException, JsonParseException
{
    if ((_numTypesValid & NR_INT) != 0) {
        _numberLong = (long) _numberInt;
    } else if ((_numTypesValid & NR_BIGINT) != 0) {
        // !!! Should check for range...
        _numberLong = _numberBigInt.longValue();
    } else if ((_numTypesValid & NR_DOUBLE) != 0) {
        // Need to check boundaries
        if (_numberDouble < MIN_LONG_D || _numberDouble > MAX_LONG_D) {
            reportOverflowLong();
        }
        _numberLong = (long) _numberDouble;
    } else if ((_numTypesValid & NR_BIGDECIMAL) != 0) {
        if (BD_MIN_LONG.compareTo(_numberBigDecimal) > 0
            || BD_MAX_LONG.compareTo(_numberBigDecimal) < 0) {
            reportOverflowLong();
        }
```

```
            _numberLong = _numberBigDecimal.longValue();
        } else {
            _throwInternal(); // should never get here
        }

        _numTypesValid |= NR_LONG;
    }

    protected void convertNumberToBigInteger()
        throws IOException, JsonParseException
    {
        if ((_numTypesValid & NR_BIGDECIMAL) != 0) {
            // here it'll just get truncated, no exceptions thrown
            _numberBigInt = _numberBigDecimal.toBigInteger();
        } else if ((_numTypesValid & NR_LONG) != 0) {
            _numberBigInt = BigInteger.valueOf(_numberLong);
        } else if ((_numTypesValid & NR_INT) != 0) {
            _numberBigInt = BigInteger.valueOf(_numberInt);
        } else if ((_numTypesValid & NR_DOUBLE) != 0) {
            _numberBigInt = BigDecimal.valueOf(_numberDouble).toBigInteger();
        } else {
            _throwInternal(); // should never get here
        }
        _numTypesValid |= NR_BIGINT;
    }

    protected void convertNumberToDouble()
        throws IOException, JsonParseException
    {
        /* 05-Aug-2008, tatus: Important note: this MUST start with
         *   more accurate representations, since we don't know which
         *   value is the original one (others get generated when
         *   requested)
         */

        if ((_numTypesValid & NR_BIGDECIMAL) != 0) {
            _numberDouble = _numberBigDecimal.doubleValue();
        } else if ((_numTypesValid & NR_BIGINT) != 0) {
            _numberDouble = _numberBigInt.doubleValue();
        } else if ((_numTypesValid & NR_LONG) != 0) {
            _numberDouble = (double) _numberLong;
        } else if ((_numTypesValid & NR_INT) != 0) {
            _numberDouble = (double) _numberInt;
        } else {
            _throwInternal(); // should never get here
        }

        _numTypesValid |= NR_DOUBLE;
```

```
    }

    protected void convertNumberToBigDecimal()
       throws IOException, JsonParseException
    {
       /* 05-Aug-2008, tatus: Important note: this MUST start with
        *   more accurate representations, since we don't know which
        *   value is the original one (others get generated when
        *   requested)
        */

       if ((_numTypesValid & NR_DOUBLE) != 0) {
          /* Let's actually parse from String representation,
           * to avoid rounding errors that non-decimal floating operations
           * would incur
           */
          _numberBigDecimal = new BigDecimal(getText());
       } else if ((_numTypesValid & NR_BIGINT) != 0) {
          _numberBigDecimal = new BigDecimal(_numberBigInt);
       } else if ((_numTypesValid & NR_LONG) != 0) {
          _numberBigDecimal = BigDecimal.valueOf(_numberLong);
       } else if ((_numTypesValid & NR_INT) != 0) {
          _numberBigDecimal = BigDecimal.valueOf((long) _numberInt);
       } else {
          _throwInternal(); // should never get here
       }
       _numTypesValid |= NR_BIGDECIMAL;
    }

    /*
    /**********************************************************
    /* Exception reporting
    /**********************************************************
     */

    protected void reportUnexpectedNumberChar(int ch, String comment)
       throws JsonParseException
    {
       String msg = "Unexpected character ("+_getCharDesc(ch)+") in numeric value";
       if (comment != null) {
          msg += ": "+comment;
       }
       _reportError(msg);
    }

    protected void reportInvalidNumber(String msg)
       throws JsonParseException
    {
```

```
        _reportError("Invalid numeric value: "+msg);
    }


    protected void reportOverflowInt()
        throws IOException, JsonParseException
    {
        _reportError("Numeric value ("+getText()+") out of range of int ("+Integer.MIN_VALUE+" -
"+Integer.MAX_VALUE+")");
    }

    protected void reportOverflowLong()
        throws IOException, JsonParseException
    {
        _reportError("Numeric value ("+getText()+") out of range of long ("+Long.MIN_VALUE+" -
"+Long.MAX_VALUE+")");
    }

}
package org.codehaus.jackson.impl;

import java.io.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.IOContext;

/**
* This is a simple low-level input reader base class, used by
* JSON parser.
* The reason for sub-classing (over composition)
* is due to need for direct access to character buffers
* and positions.
*
* @author Tatu Saloranta
*/
public abstract class ReaderBasedParserBase
    extends JsonNumericParserBase
{
    /*
    /**********************************************************
    /* Configuration
    /**********************************************************
     */

    /**
     * Reader that can be used for reading more content, if one
     * buffer from input source, but in some cases pre-loaded buffer
     * is handed to the parser.
```

```
 */
protected Reader _reader;

/*
/**********************************************************
/* Current input data
/**********************************************************
 */

/**
 * Current buffer from which data is read; generally data is read into
 * buffer from input source.
 */
protected char[] _inputBuffer;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

protected ReaderBasedParserBase(IOContext ctxt, int features, Reader r)
{
    super(ctxt, features);
    _reader = r;
    _inputBuffer = ctxt.allocTokenBuffer();
}

/*
/**********************************************************
/* Overrides
/**********************************************************
 */

@Override
public int releaseBuffered(Writer w) throws IOException
{
    int count = _inputEnd - _inputPtr;
    if (count < 1) {
        return 0;
    }
    // let's just advance ptr to end
    int origPtr = _inputPtr;
    w.write(_inputBuffer, origPtr, count);
    return count;
}

/*
```

```
    /**********************************************************
    /* Low-level reading, other
    /**********************************************************
     */

    @Override
    protected final boolean loadMore()
        throws IOException
    {
        _currInputProcessed += _inputEnd;
        _currInputRowStart -= _inputEnd;

        if (_reader != null) {
            int count = _reader.read(_inputBuffer, 0, _inputBuffer.length);
            if (count > 0) {
                _inputPtr = 0;
                _inputEnd = count;
                return true;
            }
            // End of input
            _closeInput();
            // Should never return 0, so let's fail
            if (count == 0) {
                throw new IOException("Reader returned 0 characters when trying to read "+_inputEnd);
            }
        }
        return false;
    }

    protected char getNextChar(String eofMsg)
        throws IOException, JsonParseException
    {
        if (_inputPtr >= _inputEnd) {
            if (!loadMore()) {
                _reportInvalidEOF(eofMsg);
            }
        }
        return _inputBuffer[_inputPtr++];
    }

    @Override
    protected void _closeInput() throws IOException
    {
        /* 25-Nov-2008, tatus: As per [JACKSON-16] we are not to call close()
         *   on the underlying Reader, unless we "own" it, or auto-closing
         *   feature is enabled.
         *   One downside is that when using our optimized
         *   Reader (granted, we only do that for UTF-32...) this
```

```
         *   means that buffer recycling won't work correctly.
         */
        if (_reader != null) {
            if (_ioContext.isResourceManaged() || isEnabled(Feature.AUTO_CLOSE_SOURCE)) {
                _reader.close();
            }
            _reader = null;
        }
    }

    /**
     * Method called to release internal buffers owned by the base
     * reader. This may be called along with {@link #_closeInput} (for
     * example, when explicitly closing this reader instance), or
     * separately (if need be).
     */
    @Override
    protected void _releaseBuffers()
        throws IOException
    {
        super._releaseBuffers();
        char[] buf = _inputBuffer;
        if (buf != null) {
            _inputBuffer = null;
            _ioContext.releaseTokenBuffer(buf);
        }
    }
}
package org.codehaus.jackson.impl;

import java.io.IOException;
import java.io.Reader;

import org.codehaus.jackson.io.IOContext;
import org.codehaus.jackson.JsonParseException;
import org.codehaus.jackson.JsonToken;

/**
* Intermediate class that implements handling of numeric parsing.
* Separate from the actual parser class just to isolate numeric
* parsing: would be nice to use aggregation, but unfortunately
* many parts are hard to implement without direct access to
* underlying buffers.
*/
public abstract class ReaderBasedNumericParser
    extends ReaderBasedParserBase
{
    /*
```

```
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public ReaderBasedNumericParser(IOContext pc, int features, Reader r)
{
   super(pc, features, r);
}

/*
/**********************************************************
/* Textual parsing of number values
/**********************************************************
 */

/**
 * Initial parsing method for number values. It needs to be able
 * to parse enough input to be able to determine whether the
 * value is to be considered a simple integer value, or a more
 * generic decimal value: latter of which needs to be expressed
 * as a floating point number. The basic rule is that if the number
 * has no fractional or exponential part, it is an integer; otherwise
 * a floating point number.
 *<p>
 * Because much of input has to be processed in any case, no partial
 * parsing is done: all input text will be stored for further
 * processing. However, actual numeric value conversion will be
 * deferred, since it is usually the most complicated and costliest
 * part of processing.
 */
protected final JsonToken parseNumberText(int ch)
   throws IOException, JsonParseException
{
   /* Although we will always be complete with respect to textual
    * representation (that is, all characters will be parsed),
    * actual conversion to a number is deferred. Thus, need to
    * note that no representations are valid yet
    */
   boolean negative = (ch == INT_MINUS);
   int ptr = _inputPtr;
   int startPtr = ptr-1; // to include sign/digit already read
   final int inputLen = _inputEnd;

   dummy_loop:
   do { // dummy loop, to be able to break out
      if (negative) { // need to read the next digit
         if (ptr >= _inputEnd) {
```

```
            break dummy_loop;
        }
        ch = _inputBuffer[ptr++];
        // First check: must have a digit to follow minus sign
        if (ch > INT_9 || ch < INT_0) {
            reportUnexpectedNumberChar(ch, "expected digit (0-9) to follow minus sign, for valid numeric value");
        }
        /* (note: has been checked for non-negative already, in
         * the dispatching code that determined it should be
         * a numeric value)
         */
    }

    /* First, let's see if the whole number is contained within
     * the input buffer unsplit. This should be the common case;
     * and to simplify processing, we will just reparse contents
     * in the alternative case (number split on buffer boundary)
     */

    int intLen = 1; // already got one

    // First let's get the obligatory integer part:

    int_loop:
    while (true) {
        if (ptr >= _inputEnd) {
            break dummy_loop;
        }
        ch = (int) _inputBuffer[ptr++];
        if (ch < INT_0 || ch > INT_9) {
            break int_loop;
        }
        // The only check: no leading zeroes
        if (++intLen == 2) { // To ensure no leading zeroes
            if (_inputBuffer[ptr-2] == '0') {
                reportInvalidNumber("Leading zeroes not allowed");
            }
        }
    }

    int fractLen = 0;

    // And then see if we get other parts
    if (ch == INT_DECIMAL_POINT) { // yes, fraction
        fract_loop:
        while (true) {
            if (ptr >= inputLen) {
                break dummy_loop;
```

```
            }
            ch = (int) _inputBuffer[ptr++];
            if (ch < INT_0 || ch > INT_9) {
                break fract_loop;
            }
            ++fractLen;
        }
        // must be followed by sequence of ints, one minimum
        if (fractLen == 0) {
            reportUnexpectedNumberChar(ch, "Decimal point not followed by a digit");
        }
    }

    int expLen = 0;
    if (ch == INT_e || ch == INT_E) { // and/or expontent
        if (ptr >= inputLen) {
            break dummy_loop;
        }
        // Sign indicator?
        ch = (int) _inputBuffer[ptr++];
        if (ch == INT_MINUS || ch == INT_PLUS) { // yup, skip for now
            if (ptr >= inputLen) {
                break dummy_loop;
            }
            ch = (int) _inputBuffer[ptr++];
        }
        while (ch <= INT_9 && ch >= INT_0) {
            ++expLen;
            if (ptr >= inputLen) {
                break dummy_loop;
            }
            ch = (int) _inputBuffer[ptr++];
        }
        // must be followed by sequence of ints, one minimum
        if (expLen == 0) {
            reportUnexpectedNumberChar(ch, "Exponent indicator not followed by a digit");
        }
    }

    // Got it all: let's add to text buffer for parsing, access
    --ptr; // need to push back following separator
    _inputPtr = ptr;
    int len = ptr-startPtr;
    _textBuffer.resetWithShared(_inputBuffer, startPtr, len);
    return reset(negative, intLen, fractLen, expLen);
} while (false);

_inputPtr = negative ? (startPtr+1) : startPtr;
```

```
    return parseNumberText2(negative);
}

/**
 * Method called to parse a number, when the primary parse
 * method has failed to parse it, due to it being split on
 * buffer boundary. As a result code is very similar, except
 * that it has to explicitly copy contents to the text buffer
 * instead of just sharing the main input buffer.
 */
private final JsonToken parseNumberText2(boolean negative)
    throws IOException, JsonParseException
{
    char[] outBuf = _textBuffer.emptyAndGetCurrentSegment();
    int outPtr = 0;

    // Need to prepend sign?
    if (negative) {
        outBuf[outPtr++] = '-';
    }

    char c;
    int intLen = 0;
    boolean eof = false;

    // Ok, first the obligatory integer part:
    int_loop:
    while (true) {
        if (_inputPtr >= _inputEnd && !loadMore()) {
            // EOF is legal for main level int values
            c = CHAR_NULL;
            eof = true;
            break int_loop;
        }
        c = _inputBuffer[_inputPtr++];
        if (c < INT_0 || c > INT_9) {
            break int_loop;
        }
        ++intLen;
        // Quickie check: no leading zeroes allowed
        if (intLen == 2) {
            if (outBuf[outPtr-1] == '0') {
                reportInvalidNumber("Leading zeroes not allowed");
            }
        }
        if (outPtr >= outBuf.length) {
            outBuf = _textBuffer.finishCurrentSegment();
            outPtr = 0;
```

```
    }
    outBuf[outPtr++] = c;
}
// Also, integer part is not optional
if (intLen == 0) {
    reportInvalidNumber("Missing integer part (next char "+_getCharDesc(c)+")");
}

int fractLen = 0;
// And then see if we get other parts
if (c == '.') { // yes, fraction
    outBuf[outPtr++] = c;

    fract_loop:
    while (true) {
        if (_inputPtr >= _inputEnd && !loadMore()) {
            eof = true;
            break fract_loop;
        }
        c = _inputBuffer[_inputPtr++];
        if (c < INT_0 || c > INT_9) {
            break fract_loop;
        }
        ++fractLen;
        if (outPtr >= outBuf.length) {
            outBuf = _textBuffer.finishCurrentSegment();
            outPtr = 0;
        }
        outBuf[outPtr++] = c;
    }
    // must be followed by sequence of ints, one minimum
    if (fractLen == 0) {
        reportUnexpectedNumberChar(c, "Decimal point not followed by a digit");
    }
}

int expLen = 0;
if (c == 'e' || c == 'E') { // exponent?
    if (outPtr >= outBuf.length) {
        outBuf = _textBuffer.finishCurrentSegment();
        outPtr = 0;
    }
    outBuf[outPtr++] = c;
    // Not optional, can require that we get one more char
    c = (_inputPtr < _inputEnd) ? _inputBuffer[_inputPtr++]
        : getNextChar("expected a digit for number exponent");
    // Sign indicator?
    if (c == '-' || c == '+') {
```

```
            if (outPtr >= outBuf.length) {
                outBuf = _textBuffer.finishCurrentSegment();
                outPtr = 0;
            }
            outBuf[outPtr++] = c;
            // Likewise, non optional:
            c = (_inputPtr < _inputEnd) ? _inputBuffer[_inputPtr++]
                : getNextChar("expected a digit for number exponent");
        }

        exp_loop:
        while (c <= INT_9 && c >= INT_0) {
            ++expLen;
            if (outPtr >= outBuf.length) {
                outBuf = _textBuffer.finishCurrentSegment();
                outPtr = 0;
            }
            outBuf[outPtr++] = c;
            if (_inputPtr >= _inputEnd && !loadMore()) {
                eof = true;
                break exp_loop;
            }
            c = _inputBuffer[_inputPtr++];
        }
        // must be followed by sequence of ints, one minimum
        if (expLen == 0) {
            reportUnexpectedNumberChar(c, "Exponent indicator not followed by a digit");
        }
    }

    // Ok; unless we hit end-of-input, need to push last char read back
    if (!eof) {
        --_inputPtr;
    }
    _textBuffer.setCurrentLength(outPtr);

    // And there we have it!
    return reset(negative, intLen, fractLen, expLen);
  }

}
package org.codehaus.jackson.impl;

import java.io.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.IOContext;
import org.codehaus.jackson.sym.*;
```

```
import org.codehaus.jackson.util.*;

/**
* This is a concrete implementation of {@link JsonParser}, which is
* based on a {@link java.io.InputStream} as the input source.
*/
public final class Utf8StreamParser
  extends StreamBasedParserBase
{
  final static byte BYTE_LF = (byte) '\n';

  private final static byte BYTE_0 = (byte) 0;

  private final static int[] sInputCodesUtf8 = CharTypes.getInputCodeUtf8();

  private final static int[] sInputCodesLatin1 = CharTypes.getInputCodeLatin1();

  /*
  /**********************************************************
  /* Configuration
  /**********************************************************
   */

  /**
   * Codec used for data binding when (if) requested.
   */
  protected ObjectCodec _objectCodec;

  /**
   * Symbol table that contains field names encountered so far
   */
  final protected BytesToNameCanonicalizer _symbols;

  /*
  /**********************************************************
  /* Parsing state
  /**********************************************************
   */

  /**
   * Temporary buffer used for name parsing.
   */
  protected int[] _quadBuffer = new int[16];

  /**
   * Flag that indicates that the current token has not yet
   * been fully processed, and needs to be finished for
   * some access (or skipped to obtain the next token)
```

```
 */
protected boolean _tokenIncomplete = false;

/**
 * Temporary storage for partially parsed name bytes.
 */
private int _quad1;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public Utf8StreamParser(IOContext ctxt, int features, InputStream in,
        ObjectCodec codec, BytesToNameCanonicalizer sym,
        byte[] inputBuffer, int start, int end,
        boolean bufferRecyclable)
{
    super(ctxt, features, in, inputBuffer, start, end, bufferRecyclable);
    _objectCodec = codec;
    _symbols = sym;
    // 12-Mar-2010, tatus: Sanity check, related to [JACKSON-259]:
    if (!JsonParser.Feature.CANONICALIZE_FIELD_NAMES.enabledIn(features)) {
        // should never construct non-canonical utf8/byte parser (instead, use Reader)
        _throwInternal();
    }
}

@Override
public ObjectCodec getCodec() {
    return _objectCodec;
}

@Override
public void setCodec(ObjectCodec c) {
    _objectCodec = c;
}

/*
/**********************************************************
/* Public API, data access
/**********************************************************
 */

@Override
public String getText()
    throws IOException, JsonParseException
```

```
{
    JsonToken t = _currToken;
    if (t == JsonToken.VALUE_STRING) {
        if (_tokenIncomplete) {
            _tokenIncomplete = false;
            _finishString(); // only strings can be incomplete
        }
        return _textBuffer.contentsAsString();
    }
    return _getText2(t);
}

protected final String _getText2(JsonToken t)
{
    if (t == null) {
        return null;
    }
    switch (t) {
    case FIELD_NAME:
        return _parsingContext.getCurrentName();

    case VALUE_STRING:
        // fall through
    case VALUE_NUMBER_INT:
    case VALUE_NUMBER_FLOAT:
        return _textBuffer.contentsAsString();
    }
    return t.asString();
}

@Override
public char[] getTextCharacters()
    throws IOException, JsonParseException
{
    if (_currToken != null) { // null only before/after document
        switch (_currToken) {

        case FIELD_NAME:
            if (!_nameCopied) {
                String name = _parsingContext.getCurrentName();
                int nameLen = name.length();
                if (_nameCopyBuffer == null) {
                    _nameCopyBuffer = _ioContext.allocNameCopyBuffer(nameLen);
                } else if (_nameCopyBuffer.length < nameLen) {
                    _nameCopyBuffer = new char[nameLen];
                }
                name.getChars(0, nameLen, _nameCopyBuffer, 0);
                _nameCopied = true;
```

```
            }
            return _nameCopyBuffer;

        case VALUE_STRING:
            if (_tokenIncomplete) {
                _tokenIncomplete = false;
                _finishString(); // only strings can be incomplete
            }
            // fall through
        case VALUE_NUMBER_INT:
        case VALUE_NUMBER_FLOAT:
            return _textBuffer.getTextBuffer();

        default:
            return _currToken.asCharArray();
        }
    }
    return null;
}

@Override
public int getTextLength()
    throws IOException, JsonParseException
{
    if (_currToken != null) { // null only before/after document
        switch (_currToken) {

        case FIELD_NAME:
            return _parsingContext.getCurrentName().length();
        case VALUE_STRING:
            if (_tokenIncomplete) {
                _tokenIncomplete = false;
                _finishString(); // only strings can be incomplete
            }
            // fall through
        case VALUE_NUMBER_INT:
        case VALUE_NUMBER_FLOAT:
            return _textBuffer.size();

        default:
            return _currToken.asCharArray().length;
        }
    }
    return 0;
}

@Override
public int getTextOffset() throws IOException, JsonParseException
```

```
    {
        // Most have offset of 0, only some may have other values:
        if (_currToken != null) {
            switch (_currToken) {
            case FIELD_NAME:
                return 0;
            case VALUE_STRING:
                if (_tokenIncomplete) {
                    _tokenIncomplete = false;
                    _finishString(); // only strings can be incomplete
                }
                // fall through
            case VALUE_NUMBER_INT:
            case VALUE_NUMBER_FLOAT:
                return _textBuffer.getTextOffset();
            }
        }
        return 0;
    }

    @Override
    public byte[] getBinaryValue(Base64Variant b64variant)
        throws IOException, JsonParseException
    {
        if (_currToken != JsonToken.VALUE_STRING &&
                (_currToken != JsonToken.VALUE_EMBEDDED_OBJECT || _binaryValue == null)) {
            _reportError("Current token ("+_currToken+") not VALUE_STRING or VALUE_EMBEDDED_OBJECT,
can not access as binary");
        }
        /* To ensure that we won't see inconsistent data, better clear up
         * state...
         */
        if (_tokenIncomplete) {
            try {
                _binaryValue = _decodeBase64(b64variant);
            } catch (IllegalArgumentException iae) {
                throw _constructError("Failed to decode VALUE_STRING as base64 ("+b64variant+"):
"+iae.getMessage());
            }
            /* let's clear incomplete only now; allows for accessing other
             * textual content in error cases
             */
            _tokenIncomplete = false;
        }
        return _binaryValue;
    }

    /*
```

```
/**********************************************************
/* Public API, traversal
/**********************************************************
 */

/**
 * @return Next token from the stream, if any found, or null
 *   to indicate end-of-input
 */
@Override
public JsonToken nextToken()
   throws IOException, JsonParseException
{
   /* First: field names are special -- we will always tokenize
    * (part of) value along with field name to simplify
    * state handling. If so, can and need to use secondary token:
    */
   if (_currToken == JsonToken.FIELD_NAME) {
      return _nextAfterName();
   }
   if (_tokenIncomplete) {
      _skipString(); // only strings can be partial
   }

   int i = _skipWSOrEnd();
   if (i < 0) { // end-of-input
      /* 19-Feb-2009, tatu: Should actually close/release things
       *    like input source, symbol table and recyclable buffers now.
       */
      close();
      return (_currToken = null);
   }

   /* First, need to ensure we know the starting location of token
    * after skipping leading white space
    */
   _tokenInputTotal = _currInputProcessed + _inputPtr - 1;
   _tokenInputRow = _currInputRow;
   _tokenInputCol = _inputPtr - _currInputRowStart - 1;

   // finally: clear any data retained so far
   _binaryValue = null;

   // Closing scope?
   if (i == INT_RBRACKET) {
      if (!_parsingContext.inArray()) {
         _reportMismatchedEndMarker(i, '}');
      }
```

```
            _parsingContext = _parsingContext.getParent();
            return (_currToken = JsonToken.END_ARRAY);
        }
        if (i == INT_RCURLY) {
            if (!_parsingContext.inObject()) {
                _reportMismatchedEndMarker(i, ']');
            }
            _parsingContext = _parsingContext.getParent();
            return (_currToken = JsonToken.END_OBJECT);
        }

        // Nope: do we then expect a comma?
        if (_parsingContext.expectComma()) {
            if (i != INT_COMMA) {
                _reportUnexpectedChar(i, "was expecting comma to separate "+_parsingContext.getTypeDesc()+"
entries");
            }
            i = _skipWS();
        }

        /* And should we now have a name? Always true for
         * Object contexts, since the intermediate 'expect-value'
         * state is never retained.
         */
        if (!_parsingContext.inObject()) {
            return _nextTokenNotInObject(i);
        }
        // So first parse the field name itself:
        Name n = _parseFieldName(i);
        _parsingContext.setCurrentName(n.getName());
        _currToken = JsonToken.FIELD_NAME;
        i = _skipWS();
        if (i != INT_COLON) {
            _reportUnexpectedChar(i, "was expecting a colon to separate field name and value");
        }
        i = _skipWS();

        // Ok: we must have a value... what is it? Strings are very common, check first:
        if (i == INT_QUOTE) {
            _tokenIncomplete = true;
            _nextToken = JsonToken.VALUE_STRING;
            return _currToken;
        }
        JsonToken t;

        switch (i) {
        case INT_LBRACKET:
            t = JsonToken.START_ARRAY;
```

```
            break;
        case INT_LCURLY:
            t = JsonToken.START_OBJECT;
            break;
        case INT_RBRACKET:
        case INT_RCURLY:
            // Error: neither is valid at this point; valid closers have
            // been handled earlier
            _reportUnexpectedChar(i, "expected a value");
        case INT_t:
            _matchToken(JsonToken.VALUE_TRUE);
            t = JsonToken.VALUE_TRUE;
            break;
        case INT_f:
            _matchToken(JsonToken.VALUE_FALSE);
             t = JsonToken.VALUE_FALSE;
            break;
        case INT_n:
            _matchToken(JsonToken.VALUE_NULL);
            t = JsonToken.VALUE_NULL;
            break;

        case INT_MINUS:
            /* Should we have separate handling for plus? Although
             * it is not allowed per se, it may be erroneously used,
             * and could be indicate by a more specific error message.
             */
        case INT_0:
        case INT_1:
        case INT_2:
        case INT_3:
        case INT_4:
        case INT_5:
        case INT_6:
        case INT_7:
        case INT_8:
        case INT_9:
            t = parseNumberText(i);
            break;
        default:
            t = _handleUnexpectedValue(i);
        }
        _nextToken = t;
        return _currToken;
    }

    private final JsonToken _nextTokenNotInObject(int i)
        throws IOException, JsonParseException
```

```
{
    if (i == INT_QUOTE) {
        _tokenIncomplete = true;
        return (_currToken = JsonToken.VALUE_STRING);
    }
    switch (i) {
    case INT_LBRACKET:
        _parsingContext = _parsingContext.createChildArrayContext(_tokenInputRow, _tokenInputCol);
        return (_currToken = JsonToken.START_ARRAY);
    case INT_LCURLY:
        _parsingContext = _parsingContext.createChildObjectContext(_tokenInputRow, _tokenInputCol);
        return (_currToken = JsonToken.START_OBJECT);
    case INT_RBRACKET:
    case INT_RCURLY:
        // Error: neither is valid at this point; valid closers have
        // been handled earlier
        _reportUnexpectedChar(i, "expected a value");
    case INT_t:
        _matchToken(JsonToken.VALUE_TRUE);
        return (_currToken = JsonToken.VALUE_TRUE);
    case INT_f:
        _matchToken(JsonToken.VALUE_FALSE);
        return (_currToken = JsonToken.VALUE_FALSE);
    case INT_n:
        _matchToken(JsonToken.VALUE_NULL);
        return (_currToken = JsonToken.VALUE_NULL);
    case INT_MINUS:
        /* Should we have separate handling for plus? Although
         * it is not allowed per se, it may be erroneously used,
         * and could be indicate by a more specific error message.
         */
    case INT_0:
    case INT_1:
    case INT_2:
    case INT_3:
    case INT_4:
    case INT_5:
    case INT_6:
    case INT_7:
    case INT_8:
    case INT_9:
        return (_currToken = parseNumberText(i));
    }
    return (_currToken = _handleUnexpectedValue(i));
}

private final JsonToken _nextAfterName()
{
```

```
    _nameCopied = false; // need to invalidate if it was copied
    JsonToken t = _nextToken;
    _nextToken = null;
    // Also: may need to start new context?
    if (t == JsonToken.START_ARRAY) {
        _parsingContext = _parsingContext.createChildArrayContext(_tokenInputRow, _tokenInputCol);
    } else if (t == JsonToken.START_OBJECT) {
        _parsingContext = _parsingContext.createChildObjectContext(_tokenInputRow, _tokenInputCol);
    }
    return (_currToken = t);
}

@Override
public void close() throws IOException
{
    super.close();
    // Merge found symbols, if any:
    _symbols.release();
}

/*
/**********************************************************
/* Internal methods, number parsing
/* (note: in 1.6 and prior, part of "Utf8NumericParser"
/**********************************************************
 */

/**
 * Initial parsing method for number values. It needs to be able
 * to parse enough input to be able to determine whether the
 * value is to be considered a simple integer value, or a more
 * generic decimal value: latter of which needs to be expressed
 * as a floating point number. The basic rule is that if the number
 * has no fractional or exponential part, it is an integer; otherwise
 * a floating point number.
 *<p>
 * Because much of input has to be processed in any case, no partial
 * parsing is done: all input text will be stored for further
 * processing. However, actual numeric value conversion will be
 * deferred, since it is usually the most complicated and costliest
 * part of processing.
 */
protected final JsonToken parseNumberText(int c)
    throws IOException, JsonParseException
{
    char[] outBuf = _textBuffer.emptyAndGetCurrentSegment();
    int outPtr = 0;
    boolean negative = (c == INT_MINUS);
```

```
// Need to prepend sign?
if (negative) {
   outBuf[outPtr++] = '-';
   // Must have something after sign too
   if (_inputPtr >= _inputEnd) {
      loadMoreGuaranteed();
   }
   c = (int) _inputBuffer[_inputPtr++] & 0xFF;
   // Note: must be followed by a digit
   if (c < INT_0 || c > INT_9) {
      reportInvalidNumber("Missing integer part (next char "+_getCharDesc(c)+")");
   }
}

// One special case: if first char is 0, must not be followed by a digit
if (c == INT_0) {
   _verifyNoLeadingZeroes();
}

// Ok: we can first just add digit we saw first:
outBuf[outPtr++] = (char) c;
int intLen = 1;

// And then figure out how far we can read without further checks:
int end = _inputPtr + outBuf.length;
if (end > _inputEnd) {
   end = _inputEnd;
}

// With this, we have a nice and tight loop:
while (true) {
   if (_inputPtr >= end) {
      // Long enough to be split across boundary, so:
      return _parserNumber2(outBuf, outPtr, negative, intLen);
   }
   c = (int) _inputBuffer[_inputPtr++] & 0xFF;
   if (c < INT_0 || c > INT_9) {
      break;
   }
   ++intLen;
   outBuf[outPtr++] = (char) c;
}
if (c == '.' || c == 'e' || c == 'E') {
   return _parseFloatText(outBuf, outPtr, c, negative, intLen);
}

--_inputPtr; // to push back trailing char (comma etc)
```

```
    _textBuffer.setCurrentLength(outPtr);

    // And there we have it!
    return resetInt(negative, intLen);
}

/**
 * Method called to handle parsing when input is split across buffer boundary
 * (or output is longer than segment used to store it)
 */
private final JsonToken _parserNumber2(char[] outBuf, int outPtr, boolean negative,
        int intPartLength)
    throws IOException, JsonParseException
{
    // Ok, parse the rest
    while (true) {
        if (_inputPtr >= _inputEnd && !loadMore()) {
            _textBuffer.setCurrentLength(outPtr);
            return resetInt(negative, intPartLength);
        }
        int c = (int) _inputBuffer[_inputPtr++] & 0xFF;
        if (c > INT_9 || c < INT_0) {
            if (c == '.' || c == 'e' || c == 'E') {
                return _parseFloatText(outBuf, outPtr, c, negative, intPartLength);
            }
            break;
        }
        if (outPtr >= outBuf.length) {
            outBuf = _textBuffer.finishCurrentSegment();
            outPtr = 0;
        }
        outBuf[outPtr++] = (char) c;
        ++intPartLength;
    }
    --_inputPtr; // to push back trailing char (comma etc)
    _textBuffer.setCurrentLength(outPtr);

    // And there we have it!
    return resetInt(negative, intPartLength);

}

/**
 * Method called when we have seen one zero, and want to ensure
 * it is not followed by another
 */
private final void _verifyNoLeadingZeroes()
    throws IOException, JsonParseException
```

```
{
    // Ok to have plain "0"
    if (_inputPtr >= _inputEnd && !loadMore()) {
        return;
    }
    if (_inputBuffer[_inputPtr] == BYTE_0) {
        reportInvalidNumber("Leading zeroes not allowed");
    }
}

private final JsonToken _parseFloatText(char[] outBuf, int outPtr, int c,
        boolean negative, int integerPartLength)
    throws IOException, JsonParseException
{
    int fractLen = 0;
    boolean eof = false;

    // And then see if we get other parts
    if (c == '.') { // yes, fraction
        outBuf[outPtr++] = (char) c;

        fract_loop:
        while (true) {
            if (_inputPtr >= _inputEnd && !loadMore()) {
                eof = true;
                break fract_loop;
            }
            c = (int) _inputBuffer[_inputPtr++] & 0xFF;
            if (c < INT_0 || c > INT_9) {
                break fract_loop;
            }
            ++fractLen;
            if (outPtr >= outBuf.length) {
                outBuf = _textBuffer.finishCurrentSegment();
                outPtr = 0;
            }
            outBuf[outPtr++] = (char) c;
        }
        // must be followed by sequence of ints, one minimum
        if (fractLen == 0) {
            reportUnexpectedNumberChar(c, "Decimal point not followed by a digit");
        }
    }

    int expLen = 0;
    if (c == 'e' || c == 'E') { // exponent?
        if (outPtr >= outBuf.length) {
            outBuf = _textBuffer.finishCurrentSegment();
```

```
            outPtr = 0;
        }
        outBuf[outPtr++] = (char) c;
        // Not optional, can require that we get one more char
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        c = (int) _inputBuffer[_inputPtr++] & 0xFF;
        // Sign indicator?
        if (c == '-' || c == '+') {
            if (outPtr >= outBuf.length) {
                outBuf = _textBuffer.finishCurrentSegment();
                outPtr = 0;
            }
            outBuf[outPtr++] = (char) c;
            // Likewise, non optional:
            if (_inputPtr >= _inputEnd) {
                loadMoreGuaranteed();
            }
            c = (int) _inputBuffer[_inputPtr++] & 0xFF;
        }

        exp_loop:
        while (c <= INT_9 && c >= INT_0) {
            ++expLen;
            if (outPtr >= outBuf.length) {
                outBuf = _textBuffer.finishCurrentSegment();
                outPtr = 0;
            }
            outBuf[outPtr++] = (char) c;
            if (_inputPtr >= _inputEnd && !loadMore()) {
                eof = true;
                break exp_loop;
            }
            c = (int) _inputBuffer[_inputPtr++] & 0xFF;
        }
        // must be followed by sequence of ints, one minimum
        if (expLen == 0) {
            reportUnexpectedNumberChar(c, "Exponent indicator not followed by a digit");
        }
    }

    // Ok; unless we hit end-of-input, need to push last char read back
    if (!eof) {
        --_inputPtr;
    }
    _textBuffer.setCurrentLength(outPtr);
```

```
    // And there we have it!
    return resetFloat(negative, integerPartLength, fractLen, expLen);
}

/*
/**********************************************************
/* Internal methods, secondary parsing
/**********************************************************
 */

protected final Name _parseFieldName(int i)
    throws IOException, JsonParseException
{
    if (i != INT_QUOTE) {
        return _handleUnusualFieldName(i);
    }
    // First: can we optimize out bounds checks?
    if ((_inputPtr + 9) > _inputEnd) { // Need 8 chars, plus one trailing (quote)
        return slowParseFieldName();
    }

    // If so, can also unroll loops nicely
    /* 25-Nov-2008, tatu: This may seem weird, but here we do
     *   NOT want to worry about UTF-8 decoding. Rather, we'll
     *   assume that part is ok (if not it will get caught
     *   later on), and just handle quotes and backslashes here.
     */
    final byte[] input = _inputBuffer;
    final int[] codes = sInputCodesLatin1;

    int q = input[_inputPtr++] & 0xFF;

    if (codes[q] == 0) {
        i = input[_inputPtr++] & 0xFF;
        if (codes[i] == 0) {
            q = (q << 8) | i;
            i = input[_inputPtr++] & 0xFF;
            if (codes[i] == 0) {
                q = (q << 8) | i;
                i = input[_inputPtr++] & 0xFF;
                if (codes[i] == 0) {
                    q = (q << 8) | i;
                    i = input[_inputPtr++] & 0xFF;
                    if (codes[i] == 0) {
                        _quad1 = q;
                        return parseMediumFieldName(i, codes);
                    }
                    if (i == INT_QUOTE) { // one byte/char case or broken
```

```
                return findName(q, 4);
            }
            return parseFieldName(q, i, 4);
        }
        if (i == INT_QUOTE) { // one byte/char case or broken
            return findName(q, 3);
        }
        return parseFieldName(q, i, 3);
    }
    if (i == INT_QUOTE) { // one byte/char case or broken
        return findName(q, 2);
    }
    return parseFieldName(q, i, 2);
}
if (i == INT_QUOTE) { // one byte/char case or broken
    return findName(q, 1);
}
return parseFieldName(q, i, 1);
}
if (q == INT_QUOTE) { // special case, ""
    return BytesToNameCanonicalizer.getEmptyName();
}
return parseFieldName(0, q, 0); // quoting or invalid char
}

protected final Name parseMediumFieldName(int q2, final int[] codes)
    throws IOException, JsonParseException
{
    // Ok, got 5 name bytes so far
    int i = _inputBuffer[_inputPtr++] & 0xFF;
    if (codes[i] != 0) {
        if (i == INT_QUOTE) { // 5 bytes
            return findName(_quad1, q2, 1);
        }
        return parseFieldName(_quad1, q2, i, 1); // quoting or invalid char
    }
    q2 = (q2 << 8) | i;
    i = _inputBuffer[_inputPtr++] & 0xFF;
    if (codes[i] != 0) {
        if (i == INT_QUOTE) { // 6 bytes
            return findName(_quad1, q2, 2);
        }
        return parseFieldName(_quad1, q2, i, 2);
    }
    q2 = (q2 << 8) | i;
    i = _inputBuffer[_inputPtr++] & 0xFF;
    if (codes[i] != 0) {
        if (i == INT_QUOTE) { // 7 bytes
```

```
            return findName(_quad1, q2, 3);
        }
        return parseFieldName(_quad1, q2, i, 3);
    }
    q2 = (q2 << 8) | i;
    i = _inputBuffer[_inputPtr++] & 0xFF;
    if (codes[i] != 0) {
        if (i == INT_QUOTE) { // 8 bytes
            return findName(_quad1, q2, 4);
        }
        return parseFieldName(_quad1, q2, i, 4);
    }
    _quadBuffer[0] = _quad1;
    _quadBuffer[1] = q2;
    return parseLongFieldName(i);
}

protected Name parseLongFieldName(int q)
    throws IOException, JsonParseException
{
    // As explained above, will ignore utf-8 encoding at this point
    final int[] codes = sInputCodesLatin1;
    int qlen = 2;

    while (true) {
        /* Let's offline if we hit buffer boundary (otherwise would
         * need to [try to] align input, which is bit complicated
         * and may not always be possible)
         */
        if ((_inputEnd - _inputPtr) < 4) {
            return parseEscapedFieldName(_quadBuffer, qlen, 0, q, 0);
        }
        // Otherwise can skip boundary checks for 4 bytes in loop

        int i = _inputBuffer[_inputPtr++] & 0xFF;
        if (codes[i] != 0) {
            if (i == INT_QUOTE) {
                return findName(_quadBuffer, qlen, q, 1);
            }
            return parseEscapedFieldName(_quadBuffer, qlen, q, i, 1);
        }

        q = (q << 8) | i;
        i = _inputBuffer[_inputPtr++] & 0xFF;
        if (codes[i] != 0) {
            if (i == INT_QUOTE) {
                return findName(_quadBuffer, qlen, q, 2);
            }
```

```
            return parseEscapedFieldName(_quadBuffer, qlen, q, i, 2);
        }

        q = (q << 8) | i;
        i = _inputBuffer[_inputPtr++] & 0xFF;
        if (codes[i] != 0) {
            if (i == INT_QUOTE) {
                return findName(_quadBuffer, qlen, q, 3);
            }
            return parseEscapedFieldName(_quadBuffer, qlen, q, i, 3);
        }

        q = (q << 8) | i;
        i = _inputBuffer[_inputPtr++] & 0xFF;
        if (codes[i] != 0) {
            if (i == INT_QUOTE) {
                return findName(_quadBuffer, qlen, q, 4);
            }
            return parseEscapedFieldName(_quadBuffer, qlen, q, i, 4);
        }

        // Nope, no end in sight. Need to grow quad array etc
        if (qlen >= _quadBuffer.length) {
            _quadBuffer = growArrayBy(_quadBuffer, qlen);
        }
        _quadBuffer[qlen++] = q;
        q = i;
    }
}

/**
 * Method called when not even first 8 bytes are guaranteed
 * to come consequtively. Happens rarely, so this is offlined;
 * plus we'll also do full checks for escaping etc.
 */
protected Name slowParseFieldName()
    throws IOException, JsonParseException
{
    if (_inputPtr >= _inputEnd) {
        if (!loadMore()) {
            _reportInvalidEOF(": was expecting closing '\"' for name");
        }
    }
    int i = _inputBuffer[_inputPtr++] & 0xFF;
    if (i == INT_QUOTE) { // special case, ""
        return BytesToNameCanonicalizer.getEmptyName();
    }
    return parseEscapedFieldName(_quadBuffer, 0, 0, i, 0);
```

```
}

private final Name parseFieldName(int q1, int ch, int lastQuadBytes)
   throws IOException, JsonParseException
{
   return parseEscapedFieldName(_quadBuffer, 0, q1, ch, lastQuadBytes);
}

private final Name parseFieldName(int q1, int q2, int ch, int lastQuadBytes)
   throws IOException, JsonParseException
{
   _quadBuffer[0] = q1;
   return parseEscapedFieldName(_quadBuffer, 1, q2, ch, lastQuadBytes);
}

/**
 * Slower parsing method which is generally branched to when
 * an escape sequence is detected (or alternatively for long
 * names, or ones crossing input buffer boundary). In any case,
 * needs to be able to handle more exceptional cases, gets
 * slower, and hance is offlined to a separate method.
 */
protected Name parseEscapedFieldName(int[] quads, int qlen, int currQuad, int ch,
                                     int currQuadBytes)
   throws IOException, JsonParseException
{
   /* 25-Nov-2008, tatu: This may seem weird, but here we do
    *   NOT want to worry about UTF-8 decoding. Rather, we'll
    *   assume that part is ok (if not it will get caught
    *   later on), and just handle quotes and backslashes here.
    */
   final int[] codes = sInputCodesLatin1;

   while (true) {
      if (codes[ch] != 0) {
         if (ch == INT_QUOTE) { // we are done
            break;
         }
         // Unquoted white space?
         if (ch != INT_BACKSLASH) {
            // As per [JACKSON-208], call can now return:
            _throwUnquotedSpace(ch, "name");
         } else {
            // Nope, escape sequence
            ch = _decodeEscaped();
         }
         /* Oh crap. May need to UTF-8 (re-)encode it, if it's
          * beyond 7-bit ascii. Gets pretty messy.
```

```
         * If this happens often, may want to use different name
         * canonicalization to avoid these hits.
         */
        if (ch > 127) {
            // Ok, we'll need room for first byte right away
            if (currQuadBytes >= 4) {
                if (qlen >= quads.length) {
                    _quadBuffer = quads = growArrayBy(quads, quads.length);
                }
                quads[qlen++] = currQuad;
                currQuad = 0;
                currQuadBytes = 0;
            }
            if (ch < 0x800) { // 2-byte
                currQuad = (currQuad << 8) | (0xc0 | (ch >> 6));
                ++currQuadBytes;
                // Second byte gets output below:
            } else { // 3 bytes; no need to worry about surrogates here
                currQuad = (currQuad << 8) | (0xe0 | (ch >> 12));
                ++currQuadBytes;
                // need room for middle byte?
                if (currQuadBytes >= 4) {
                    if (qlen >= quads.length) {
                        _quadBuffer = quads = growArrayBy(quads, quads.length);
                    }
                    quads[qlen++] = currQuad;
                    currQuad = 0;
                    currQuadBytes = 0;
                }
                currQuad = (currQuad << 8) | (0x80 | ((ch >> 6) & 0x3f));
                ++currQuadBytes;
            }
            // And same last byte in both cases, gets output below:
            ch = 0x80 | (ch & 0x3f);
        }
    }
    // Ok, we have one more byte to add at any rate:
    if (currQuadBytes < 4) {
        ++currQuadBytes;
        currQuad = (currQuad << 8) | ch;
    } else {
        if (qlen >= quads.length) {
            _quadBuffer = quads = growArrayBy(quads, quads.length);
        }
        quads[qlen++] = currQuad;
        currQuad = ch;
        currQuadBytes = 1;
    }
```

```
            if (_inputPtr >= _inputEnd) {
                if (!loadMore()) {
                    _reportInvalidEOF(" in field name");
                }
            }
            ch = _inputBuffer[_inputPtr++] & 0xFF;
        }

        if (currQuadBytes > 0) {
            if (qlen >= quads.length) {
                _quadBuffer = quads = growArrayBy(quads, quads.length);
            }
            quads[qlen++] = currQuad;
        }
        Name name = _symbols.findName(quads, qlen);
        if (name == null) {
            name = addName(quads, qlen, currQuadBytes);
        }
        return name;
    }

    /**
     * Method called when we see non-white space character other
     * than double quote, when expecting a field name.
     * In standard mode will just throw an expection; but
     * in non-standard modes may be able to parse name.
     */
    protected final Name _handleUnusualFieldName(int ch)
        throws IOException, JsonParseException
    {
        // [JACKSON-173]: allow single quotes
        if (ch == INT_APOSTROPHE && isEnabled(Feature.ALLOW_SINGLE_QUOTES)) {
            return _parseApostropheFieldName();
        }
        // [JACKSON-69]: allow unquoted names if feature enabled:
        if (!isEnabled(Feature.ALLOW_UNQUOTED_FIELD_NAMES)) {
            _reportUnexpectedChar(ch, "was expecting double-quote to start field name");
        }
        /* Also: note that although we use a different table here,
         * it does NOT handle UTF-8 decoding. It'll just pass those
         * high-bit codes as acceptable for later decoding.
         */
        final int[] codes = CharTypes.getInputCodeUtf8JsNames();
        // Also: must start with a valid character...
        if (codes[ch] != 0) {
            _reportUnexpectedChar(ch, "was expecting either valid name character (for unquoted name) or double-quote
(for quoted) to start field name");
        }
```

```
    /* Ok, now; instead of ultra-optimizing parsing here (as with
     * regular JSON names), let's just use the generic "slow"
     * variant. Can measure its impact later on if need be
     */
    int[] quads = _quadBuffer;
    int qlen = 0;
    int currQuad = 0;
    int currQuadBytes = 0;

    while (true) {
        // Ok, we have one more byte to add at any rate:
        if (currQuadBytes < 4) {
            ++currQuadBytes;
            currQuad = (currQuad << 8) | ch;
        } else {
            if (qlen >= quads.length) {
                _quadBuffer = quads = growArrayBy(quads, quads.length);
            }
            quads[qlen++] = currQuad;
            currQuad = ch;
            currQuadBytes = 1;
        }
        if (_inputPtr >= _inputEnd) {
            if (!loadMore()) {
                _reportInvalidEOF(" in field name");
            }
        }
        ch = _inputBuffer[_inputPtr] & 0xFF;
        if (codes[ch] != 0) {
            break;
        }
        ++_inputPtr;
    }

    if (currQuadBytes > 0) {
        if (qlen >= quads.length) {
            _quadBuffer = quads = growArrayBy(quads, quads.length);
        }
        quads[qlen++] = currQuad;
    }
    Name name = _symbols.findName(quads, qlen);
    if (name == null) {
        name = addName(quads, qlen, currQuadBytes);
    }
    return name;
}
```

```
/* Parsing to support [JACKSON-173]. Plenty of duplicated code;
 * main reason being to try to avoid slowing down fast path
 * for valid JSON -- more alternatives, more code, generally
 * bit slower execution.
 */
protected final Name _parseApostropheFieldName()
    throws IOException, JsonParseException
{
    if (_inputPtr >= _inputEnd) {
        if (!loadMore()) {
            _reportInvalidEOF(": was expecting closing '\" for name");
        }
    }
    int ch = _inputBuffer[_inputPtr++] & 0xFF;
    if (ch == INT_APOSTROPHE) { // special case, "
        return BytesToNameCanonicalizer.getEmptyName();
    }
    int[] quads = _quadBuffer;
    int qlen = 0;
    int currQuad = 0;
    int currQuadBytes = 0;

    // Copied from parseEscapedFieldName, with minor mods:

    final int[] codes = sInputCodesLatin1;

    while (true) {
        if (ch == INT_APOSTROPHE) {
            break;
        }
        // additional check to skip handling of double-quotes
        if (ch != INT_QUOTE && codes[ch] != 0) {
            if (ch != INT_BACKSLASH) {
                // Unquoted white space?
                // As per [JACKSON-208], call can now return:
                _throwUnquotedSpace(ch, "name");
            } else {
                // Nope, escape sequence
                ch = _decodeEscaped();
            }
            /* Oh crap. May need to UTF-8 (re-)encode it, if it's
             * beyond 7-bit ascii. Gets pretty messy.
             * If this happens often, may want to use different name
             * canonicalization to avoid these hits.
             */
            if (ch > 127) {
                // Ok, we'll need room for first byte right away
                if (currQuadBytes >= 4) {
```

```
            if (qlen >= quads.length) {
                _quadBuffer = quads = growArrayBy(quads, quads.length);
            }
            quads[qlen++] = currQuad;
            currQuad = 0;
            currQuadBytes = 0;
        }
        if (ch < 0x800) { // 2-byte
            currQuad = (currQuad << 8) | (0xc0 | (ch >> 6));
            ++currQuadBytes;
            // Second byte gets output below:
        } else { // 3 bytes; no need to worry about surrogates here
            currQuad = (currQuad << 8) | (0xe0 | (ch >> 12));
            ++currQuadBytes;
            // need room for middle byte?
            if (currQuadBytes >= 4) {
                if (qlen >= quads.length) {
                    _quadBuffer = quads = growArrayBy(quads, quads.length);
                }
                quads[qlen++] = currQuad;
                currQuad = 0;
                currQuadBytes = 0;
            }
            currQuad = (currQuad << 8) | (0x80 | ((ch >> 6) & 0x3f));
            ++currQuadBytes;
        }
        // And same last byte in both cases, gets output below:
        ch = 0x80 | (ch & 0x3f);
    }
}
// Ok, we have one more byte to add at any rate:
if (currQuadBytes < 4) {
    ++currQuadBytes;
    currQuad = (currQuad << 8) | ch;
} else {
    if (qlen >= quads.length) {
        _quadBuffer = quads = growArrayBy(quads, quads.length);
    }
    quads[qlen++] = currQuad;
    currQuad = ch;
    currQuadBytes = 1;
}
if (_inputPtr >= _inputEnd) {
    if (!loadMore()) {
        _reportInvalidEOF(" in field name");
    }
}
ch = _inputBuffer[_inputPtr++] & 0xFF;
```

```
    }

    if (currQuadBytes > 0) {
        if (qlen >= quads.length) {
            _quadBuffer = quads = growArrayBy(quads, quads.length);
        }
        quads[qlen++] = currQuad;
    }
    Name name = _symbols.findName(quads, qlen);
    if (name == null) {
        name = addName(quads, qlen, currQuadBytes);
    }
    return name;
}

/*
/**********************************************************
/* Internal methods, symbol (name) handling
/**********************************************************
 */

private final Name findName(int q1, int lastQuadBytes)
    throws JsonParseException
{
    // Usually we'll find it from the canonical symbol table already
    Name name = _symbols.findName(q1);
    if (name != null) {
        return name;
    }
    // If not, more work. We'll need add stuff to buffer
    _quadBuffer[0] = q1;
    return addName(_quadBuffer, 1, lastQuadBytes);
}

private final Name findName(int q1, int q2, int lastQuadBytes)
    throws JsonParseException
{
    // Usually we'll find it from the canonical symbol table already
    Name name = _symbols.findName(q1, q2);
    if (name != null) {
        return name;
    }
    // If not, more work. We'll need add stuff to buffer
    _quadBuffer[0] = q1;
    _quadBuffer[1] = q2;
    return addName(_quadBuffer, 2, lastQuadBytes);
}
```

```
private final Name findName(int[] quads, int qlen, int lastQuad, int lastQuadBytes)
    throws JsonParseException
{
    if (qlen >= quads.length) {
        _quadBuffer = quads = growArrayBy(quads, quads.length);
    }
    quads[qlen++] = lastQuad;
    Name name = _symbols.findName(quads, qlen);
    if (name == null) {
        return addName(quads, qlen, lastQuadBytes);
    }
    return name;
}

/**
 * This is the main workhorse method used when we take a symbol
 * table miss. It needs to demultiplex individual bytes, decode
 * multi-byte chars (if any), and then construct Name instance
 * and add it to the symbol table.
 */
private final Name addName(int[] quads, int qlen, int lastQuadBytes)
    throws JsonParseException
{
    /* Ok: must decode UTF-8 chars. No other validation is
     * needed, since unescaping has been done earlier as necessary
     * (as well as error reporting for unescaped control chars)
     */
    // 4 bytes per quad, except last one maybe less
    int byteLen = (qlen << 2) - 4 + lastQuadBytes;

    /* And last one is not correctly aligned (leading zero bytes instead
     * need to shift a bit, instead of trailing). Only need to shift it
     * for UTF-8 decoding; need revert for storage (since key will not
     * be aligned, to optimize lookup speed)
     */
    int lastQuad;

    if (lastQuadBytes < 4) {
        lastQuad = quads[qlen-1];
        // 8/16/24 bit left shift
        quads[qlen-1] = (lastQuad << ((4 - lastQuadBytes) << 3));
    } else {
        lastQuad = 0;
    }

    // Need some working space, TextBuffer works well:
    char[] cbuf = _textBuffer.emptyAndGetCurrentSegment();
    int cix = 0;
```

```
for (int ix = 0; ix < byteLen; ) {
    int ch = quads[ix >> 2]; // current quad, need to shift+mask
    int byteIx = (ix & 3);
    ch = (ch >> ((3 - byteIx) << 3)) & 0xFF;
    ++ix;

    if (ch > 127) { // multi-byte
        int needed;
        if ((ch & 0xE0) == 0xC0) { // 2 bytes (0x0080 - 0x07FF)
            ch &= 0x1F;
            needed = 1;
        } else if ((ch & 0xF0) == 0xE0) { // 3 bytes (0x0800 - 0xFFFF)
            ch &= 0x0F;
            needed = 2;
        } else if ((ch & 0xF8) == 0xF0) { // 4 bytes; double-char with surrogates and all...
            ch &= 0x07;
            needed = 3;
        } else { // 5- and 6-byte chars not valid xml chars
            _reportInvalidInitial(ch);
            needed = ch = 1; // never really gets this far
        }
        if ((ix + needed) > byteLen) {
            _reportInvalidEOF(" in field name");
        }

        // Ok, always need at least one more:
        int ch2 = quads[ix >> 2]; // current quad, need to shift+mask
        byteIx = (ix & 3);
        ch2 = (ch2 >> ((3 - byteIx) << 3));
        ++ix;

        if ((ch2 & 0xC0) != 0x080) {
            _reportInvalidOther(ch2);
        }
        ch = (ch << 6) | (ch2 & 0x3F);
        if (needed > 1) {
            ch2 = quads[ix >> 2];
            byteIx = (ix & 3);
            ch2 = (ch2 >> ((3 - byteIx) << 3));
            ++ix;

            if ((ch2 & 0xC0) != 0x080) {
                _reportInvalidOther(ch2);
            }
            ch = (ch << 6) | (ch2 & 0x3F);
            if (needed > 2) { // 4 bytes? (need surrogates on output)
                ch2 = quads[ix >> 2];
```

```
                byteIx = (ix & 3);
                ch2 = (ch2 >> ((3 - byteIx) << 3));
                ++ix;
                if ((ch2 & 0xC0) != 0x080) {
                   _reportInvalidOther(ch2 & 0xFF);
                }
                ch = (ch << 6) | (ch2 & 0x3F);
             }
          }
          if (needed > 2) { // surrogate pair? once again, let's output one here, one later on
             ch -= 0x10000; // to normalize it starting with 0x0
             if (cix >= cbuf.length) {
                cbuf = _textBuffer.expandCurrentSegment();
             }
             cbuf[cix++] = (char) (0xD800 + (ch >> 10));
             ch = 0xDC00 | (ch & 0x03FF);
          }
       }
       if (cix >= cbuf.length) {
          cbuf = _textBuffer.expandCurrentSegment();
       }
       cbuf[cix++] = (char) ch;
    }

    /* Ok. Now we have the character array, and can construct the
     * String (as well as check proper composition of semicolons
     * for ns-aware mode...)
     */
    String baseName = new String(cbuf, 0, cix);
    // And finally, un-align if necessary
    if (lastQuadBytes < 4) {
       quads[qlen-1] = lastQuad;
    }
    return _symbols.addName(baseName, quads, qlen);
}

/*
/**********************************************************
/* Internal methods, String value parsing
/**********************************************************
 */

@Override
protected void _finishString()
    throws IOException, JsonParseException
{
    // First, single tight loop for ASCII content, not split across input buffer boundary:
    int ptr = _inputPtr;
```

```
    if (ptr >= _inputEnd) {
        loadMoreGuaranteed();
        ptr = _inputPtr;
    }
    int outPtr = 0;
    char[] outBuf = _textBuffer.emptyAndGetCurrentSegment();
    final int[] codes = sInputCodesUtf8;

    final int max = Math.min(_inputEnd, (ptr + outBuf.length));
    final byte[] inputBuffer = _inputBuffer;
    while (ptr < max) {
        int c = (int) inputBuffer[ptr] & 0xFF;
        if (codes[c] != 0) {
            if (c == INT_QUOTE) {
                _inputPtr = ptr+1;
                _textBuffer.setCurrentLength(outPtr);
                return;
            }
            break;
        }
        ++ptr;
        outBuf[outPtr++] = (char) c;
    }
    _inputPtr = ptr;
    _finishString2(outBuf, outPtr);
}

private final void _finishString2(char[] outBuf, int outPtr)
    throws IOException, JsonParseException
{
    int c;

    // Here we do want to do full decoding, hence:
    final int[] codes = sInputCodesUtf8;
    final byte[] inputBuffer = _inputBuffer;

    main_loop:
    while (true) {
        // Then the tight ASCII non-funny-char loop:
        ascii_loop:
        while (true) {
            int ptr = _inputPtr;
            if (ptr >= _inputEnd) {
                loadMoreGuaranteed();
                ptr = _inputPtr;
            }
            if (outPtr >= outBuf.length) {
                outBuf = _textBuffer.finishCurrentSegment();
```

```
            outPtr = 0;
        }
        final int max = Math.min(_inputEnd, (ptr + (outBuf.length - outPtr)));
        while (ptr < max) {
            c = (int) inputBuffer[ptr++] & 0xFF;
            if (codes[c] != 0) {
                _inputPtr = ptr;
                break ascii_loop;
            }
            outBuf[outPtr++] = (char) c;
        }
        _inputPtr = ptr;
    }
    // Ok: end marker, escape or multi-byte?
    if (c == INT_QUOTE) {
        break main_loop;
    }

    switch (codes[c]) {
    case 1: // backslash
        c = _decodeEscaped();
        break;
    case 2: // 2-byte UTF
        c = _decodeUtf8_2(c);
        break;
    case 3: // 3-byte UTF
        if ((_inputEnd - _inputPtr) >= 2) {
            c = _decodeUtf8_3fast(c);
        } else {
            c = _decodeUtf8_3(c);
        }
        break;
    case 4: // 4-byte UTF
        c = _decodeUtf8_4(c);
        // Let's add first part right away:
        outBuf[outPtr++] = (char) (0xD800 | (c >> 10));
        if (outPtr >= outBuf.length) {
            outBuf = _textBuffer.finishCurrentSegment();
            outPtr = 0;
        }
        c = 0xDC00 | (c & 0x3FF);
        // And let the other char output down below
        break;
    default:
        if (c < INT_SPACE) {
            // As per [JACKSON-208], call can now return:
            _throwUnquotedSpace(c, "string value");
        } else {
```

```
                // Is this good enough error message?
                _reportInvalidChar(c);
            }
        }
        // Need more room?
        if (outPtr >= outBuf.length) {
            outBuf = _textBuffer.finishCurrentSegment();
            outPtr = 0;
        }
        // Ok, let's add char to output:
        outBuf[outPtr++] = (char) c;
    }
    _textBuffer.setCurrentLength(outPtr);
}

/**
 * Method called to skim through rest of unparsed String value,
 * if it is not needed. This can be done bit faster if contents
 * need not be stored for future access.
 */
protected void _skipString()
    throws IOException, JsonParseException
{
    _tokenIncomplete = false;

    // Need to be fully UTF-8 aware here:
    final int[] codes = sInputCodesUtf8;
    final byte[] inputBuffer = _inputBuffer;

    main_loop:
    while (true) {
        int c;

        ascii_loop:
        while (true) {
            int ptr = _inputPtr;
            int max = _inputEnd;
            if (ptr >= max) {
                loadMoreGuaranteed();
                ptr = _inputPtr;
                max = _inputEnd;
            }
            while (ptr < max) {
                c = (int) inputBuffer[ptr++] & 0xFF;
                if (codes[c] != 0) {
                    _inputPtr = ptr;
                    break ascii_loop;
                }
```

```
            }
            _inputPtr = ptr;
        }
        // Ok: end marker, escape or multi-byte?
        if (c == INT_QUOTE) {
            break main_loop;
        }

        switch (codes[c]) {
        case 1: // backslash
            _decodeEscaped();
            break;
        case 2: // 2-byte UTF
            _skipUtf8_2(c);
            break;
        case 3: // 3-byte UTF
            _skipUtf8_3(c);
            break;
        case 4: // 4-byte UTF
            _skipUtf8_4(c);
            break;
        default:
            if (c < INT_SPACE) {
                // As per [JACKSON-208], call can now return:
                _throwUnquotedSpace(c, "string value");
            } else {
                // Is this good enough error message?
                _reportInvalidChar(c);
            }
        }
    }
}

/**
 * Method for handling cases where first non-space character
 * of an expected value token is not legal for standard JSON content.
 *
 * @since 1.3
 */
protected final JsonToken _handleUnexpectedValue(int c)
    throws IOException, JsonParseException
{
    // Most likely an error, unless we are to allow single-quote-strings
    if (c != INT_APOSTROPHE || !isEnabled(Feature.ALLOW_SINGLE_QUOTES)) {
        _reportUnexpectedChar(c, "expected a valid value (number, String, array, object, 'true', 'false' or 'null')");
    }

    // Otherwise almost verbatim copy of _finishString()
```

```
int outPtr = 0;
char[] outBuf = _textBuffer.emptyAndGetCurrentSegment();

// Here we do want to do full decoding, hence:
final int[] codes = sInputCodesUtf8;
final byte[] inputBuffer = _inputBuffer;

main_loop:
while (true) {
   // Then the tight ascii non-funny-char loop:
   ascii_loop:
   while (true) {
      if (_inputPtr >= _inputEnd) {
         loadMoreGuaranteed();
      }
      if (outPtr >= outBuf.length) {
         outBuf = _textBuffer.finishCurrentSegment();
         outPtr = 0;
      }
      int max = _inputEnd;
      {
         int max2 = _inputPtr + (outBuf.length - outPtr);
         if (max2 < max) {
            max = max2;
         }
      }
      while (_inputPtr < max) {
         c = (int) inputBuffer[_inputPtr++] & 0xFF;
         if (c == INT_APOSTROPHE || codes[c] != 0) {
            break ascii_loop;
         }
         outBuf[outPtr++] = (char) c;
      }
   }

   // Ok: end marker, escape or multi-byte?
   if (c == INT_APOSTROPHE) {
      break main_loop;
   }

   switch (codes[c]) {
   case 1: // backslash
      if (c != INT_QUOTE) { // marked as special, isn't here
         c = _decodeEscaped();
      }
      break;
   case 2: // 2-byte UTF
      c = _decodeUtf8_2(c);
```

```
            break;
        case 3: // 3-byte UTF
            if ((_inputEnd - _inputPtr) >= 2) {
                c = _decodeUtf8_3fast(c);
            } else {
                c = _decodeUtf8_3(c);
            }
            break;
        case 4: // 4-byte UTF
            c = _decodeUtf8_4(c);
            // Let's add first part right away:
            outBuf[outPtr++] = (char) (0xD800 | (c >> 10));
            if (outPtr >= outBuf.length) {
                outBuf = _textBuffer.finishCurrentSegment();
                outPtr = 0;
            }
            c = 0xDC00 | (c & 0x3FF);
            // And let the other char output down below
            break;
        default:
            if (c < INT_SPACE) {
                _throwUnquotedSpace(c, "string value");
            }
            // Is this good enough error message?
            _reportInvalidChar(c);
        }
        // Need more room?
        if (outPtr >= outBuf.length) {
            outBuf = _textBuffer.finishCurrentSegment();
            outPtr = 0;
        }
        // Ok, let's add char to output:
        outBuf[outPtr++] = (char) c;
    }
    _textBuffer.setCurrentLength(outPtr);

    return JsonToken.VALUE_STRING;
}

/*
/**********************************************************
/* Internal methods, other parsing helper methods
/**********************************************************
 */

protected void _matchToken(JsonToken token)
    throws IOException, JsonParseException
{
```

```
    // First char is already matched, need to check the rest
    byte[] matchBytes = token.asByteArray();
    int i = 1;

    for (int len = matchBytes.length; i < len; ++i) {
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        if (matchBytes[i] != _inputBuffer[_inputPtr]) {
            _reportInvalidToken(token.asString().substring(0, i));
        }
        ++_inputPtr;
    }
    /* Ok, fine; let's not bother checking anything beyond keyword.
     * If there's something wrong there, it'll cause a parsing
     * error later on.
     */
    return;
}

private void _reportInvalidToken(String matchedPart)
    throws IOException, JsonParseException
{
    StringBuilder sb = new StringBuilder(matchedPart);
    /* Let's just try to find what appears to be the token, using
     * regular Java identifier character rules. It's just a heuristic,
     * nothing fancy here (nor fast).
     */
    while (true) {
        if (_inputPtr >= _inputEnd && !loadMore()) {
            break;
        }
        int i = (int) _inputBuffer[_inputPtr++];
        char c = (char) _decodeCharForError(i);
        if (!Character.isJavaIdentifierPart(c)) {
            break;
        }
        ++_inputPtr;
        sb.append(c);
    }

    _reportError("Unrecognized token '"+sb.toString()+"': was expecting 'null', 'true' or 'false'");
}

/*
/**********************************************************
/* Internal methods, ws skipping, escape/unescape
/**********************************************************
```

```
 */

private final int _skipWS()
    throws IOException, JsonParseException
{
    while (_inputPtr < _inputEnd || loadMore()) {
        int i = _inputBuffer[_inputPtr++] & 0xFF;
        if (i > INT_SPACE) {
            if (i != INT_SLASH) {
                return i;
            }
            _skipComment();
        } else if (i != INT_SPACE) {
            if (i == INT_LF) {
                _skipLF();
            } else if (i == INT_CR) {
                _skipCR();
            } else if (i != INT_TAB) {
                _throwInvalidSpace(i);
            }
        }
    }
    throw _constructError("Unexpected end-of-input within/between "+_parsingContext.getTypeDesc()+" entries");
}

private final int _skipWSOrEnd()
    throws IOException, JsonParseException
{
    while ((_inputPtr < _inputEnd) || loadMore()) {
        int i = _inputBuffer[_inputPtr++] & 0xFF;
        if (i > INT_SPACE) {
            if (i != INT_SLASH) {
                return i;
            }
            _skipComment();
        } else if (i != INT_SPACE) {
            if (i == INT_LF) {
                _skipLF();
            } else if (i == INT_CR) {
                _skipCR();
            } else if (i != INT_TAB) {
                _throwInvalidSpace(i);
            }
        }
    }
    // We ran out of input...
    _handleEOF();
    return -1;
```

```
    }

    private final void _skipComment()
        throws IOException, JsonParseException
    {
        if (!isEnabled(Feature.ALLOW_COMMENTS)) {
            _reportUnexpectedChar('/', "maybe a (non-standard) comment? (not recognized as one since Feature
'ALLOW_COMMENTS' not enabled for parser)");
        }
        // First: check which comment (if either) it is:
        if (_inputPtr >= _inputEnd && !loadMore()) {
            _reportInvalidEOF(" in a comment");
        }
        int c = _inputBuffer[_inputPtr++] & 0xFF;
        if (c == INT_SLASH) {
            _skipCppComment();
        } else if (c == INT_ASTERISK) {
            _skipCComment();
        } else {
            _reportUnexpectedChar(c, "was expecting either '*' or '/' for a comment");
        }
    }

    private final void _skipCComment()
        throws IOException, JsonParseException
    {
        // Need to be UTF-8 aware here to decode content (for skipping)
        final int[] codes = CharTypes.getInputCodeComment();

        // Ok: need the matching '*/'
        while ((_inputPtr < _inputEnd) || loadMore()) {
            int i = (int) _inputBuffer[_inputPtr++] & 0xFF;
            int code = codes[i];
            if (code != 0) {
                switch (code) {
                case INT_ASTERISK:
                    if (_inputBuffer[_inputPtr] == INT_SLASH) {
                        ++_inputPtr;
                        return;
                    }
                    break;
                case INT_LF:
                    _skipLF();
                    break;
                case INT_CR:
                    _skipCR();
                    break;
                default: // e.g. -1
```

```
                // Is this good enough error message?
                _reportInvalidChar(i);
            }
        }
    }
    _reportInvalidEOF(" in a comment");
}

private final void _skipCppComment()
    throws IOException, JsonParseException
{
    // Ok: need to find EOF or linefeed
    final int[] codes = CharTypes.getInputCodeComment();
    while ((_inputPtr < _inputEnd) || loadMore()) {
        int i = (int) _inputBuffer[_inputPtr++] & 0xFF;
        int code = codes[i];
        if (code != 0) {
            switch (code) {
            case INT_LF:
                _skipLF();
                return;
            case INT_CR:
                _skipCR();
                return;
            case INT_ASTERISK: // nop for these comments
                break;
            default: // e.g. -1
                // Is this good enough error message?
                _reportInvalidChar(i);
            }
        }
    }
}

protected final char _decodeEscaped()
    throws IOException, JsonParseException
{
    if (_inputPtr >= _inputEnd) {
        if (!loadMore()) {
            _reportInvalidEOF(" in character escape sequence");
        }
    }
    int c = (int) _inputBuffer[_inputPtr++];

    switch ((int) c) {
        // First, ones that are mapped
    case INT_b:
        return '\b';
```

```
    case INT_t:
        return '\t';
    case INT_n:
        return '\n';
    case INT_f:
        return '\f';
    case INT_r:
        return '\r';

        // And these are to be returned as they are
    case INT_QUOTE:
    case INT_SLASH:
    case INT_BACKSLASH:
        return (char) c;

    case INT_u: // and finally hex-escaped
        break;

    default:
        return _handleUnrecognizedCharacterEscape((char) _decodeCharForError(c));
    }

    // Ok, a hex escape. Need 4 characters
    int value = 0;
    for (int i = 0; i < 4; ++i) {
        if (_inputPtr >= _inputEnd) {
            if (!loadMore()) {
                _reportInvalidEOF(" in character escape sequence");
            }
        }
        int ch = (int) _inputBuffer[_inputPtr++];
        int digit = CharTypes.charToHex(ch);
        if (digit < 0) {
            _reportUnexpectedChar(ch, "expected a hex-digit for character escape sequence");
        }
        value = (value << 4) | digit;
    }
    return (char) value;
}

protected int _decodeCharForError(int firstByte)
    throws IOException, JsonParseException
{
    int c = (int) firstByte;
    if (c < 0) { // if >= 0, is ascii and fine as is
        int needed;

        // Ok; if we end here, we got multi-byte combination
```

```
        if ((c & 0xE0) == 0xC0) { // 2 bytes (0x0080 - 0x07FF)
            c &= 0x1F;
            needed = 1;
        } else if ((c & 0xF0) == 0xE0) { // 3 bytes (0x0800 - 0xFFFF)
            c &= 0x0F;
            needed = 2;
        } else if ((c & 0xF8) == 0xF0) {
            // 4 bytes; double-char with surrogates and all...
            c &= 0x07;
            needed = 3;
        } else {
            _reportInvalidInitial(c & 0xFF);
            needed = 1; // never gets here
        }

        int d = nextByte();
        if ((d & 0xC0) != 0x080) {
            _reportInvalidOther(d & 0xFF);
        }
        c = (c << 6) | (d & 0x3F);

        if (needed > 1) { // needed == 1 means 2 bytes total
            d = nextByte(); // 3rd byte
            if ((d & 0xC0) != 0x080) {
                _reportInvalidOther(d & 0xFF);
            }
            c = (c << 6) | (d & 0x3F);
            if (needed > 2) { // 4 bytes? (need surrogates)
                d = nextByte();
                if ((d & 0xC0) != 0x080) {
                    _reportInvalidOther(d & 0xFF);
                }
                c = (c << 6) | (d & 0x3F);
            }
        }
    }
    return c;
}

/*
/**********************************************************
/* Internal methods,UTF8 decoding
/**********************************************************
 */

private final int _decodeUtf8_2(int c)
    throws IOException, JsonParseException
{
```

```
    if (_inputPtr >= _inputEnd) {
        loadMoreGuaranteed();
    }
    int d = (int) _inputBuffer[_inputPtr++];
    if ((d & 0xC0) != 0x080) {
        _reportInvalidOther(d & 0xFF, _inputPtr);
    }
    return ((c & 0x1F) << 6) | (d & 0x3F);
}

private final int _decodeUtf8_3(int c1)
    throws IOException, JsonParseException
{
    if (_inputPtr >= _inputEnd) {
        loadMoreGuaranteed();
    }
    c1 &= 0x0F;
    int d = (int) _inputBuffer[_inputPtr++];
    if ((d & 0xC0) != 0x080) {
        _reportInvalidOther(d & 0xFF, _inputPtr);
    }
    int c = (c1 << 6) | (d & 0x3F);
    if (_inputPtr >= _inputEnd) {
        loadMoreGuaranteed();
    }
    d = (int) _inputBuffer[_inputPtr++];
    if ((d & 0xC0) != 0x080) {
        _reportInvalidOther(d & 0xFF, _inputPtr);
    }
    c = (c << 6) | (d & 0x3F);
    return c;
}

private final int _decodeUtf8_3fast(int c1)
    throws IOException, JsonParseException
{
    c1 &= 0x0F;
    int d = (int) _inputBuffer[_inputPtr++];
    if ((d & 0xC0) != 0x080) {
        _reportInvalidOther(d & 0xFF, _inputPtr);
    }
    int c = (c1 << 6) | (d & 0x3F);
    d = (int) _inputBuffer[_inputPtr++];
    if ((d & 0xC0) != 0x080) {
        _reportInvalidOther(d & 0xFF, _inputPtr);
    }
    c = (c << 6) | (d & 0x3F);
    return c;
```

```
    }

    /**
     * @return Character value <b>minus 0x10000</c>; this so that caller
     *    can readily expand it to actual surrogates
     */
    private final int _decodeUtf8_4(int c)
        throws IOException, JsonParseException
    {
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        int d = (int) _inputBuffer[_inputPtr++];
        if ((d & 0xC0) != 0x080) {
            _reportInvalidOther(d & 0xFF, _inputPtr);
        }
        c = ((c & 0x07) << 6) | (d & 0x3F);

        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        d = (int) _inputBuffer[_inputPtr++];
        if ((d & 0xC0) != 0x080) {
            _reportInvalidOther(d & 0xFF, _inputPtr);
        }
        c = (c << 6) | (d & 0x3F);
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        d = (int) _inputBuffer[_inputPtr++];
        if ((d & 0xC0) != 0x080) {
            _reportInvalidOther(d & 0xFF, _inputPtr);
        }

        /* note: won't change it to negative here, since caller
         * already knows it'll need a surrogate
         */
        return ((c << 6) | (d & 0x3F)) - 0x10000;
    }

    private final void _skipUtf8_2(int c)
        throws IOException, JsonParseException
    {
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        c = (int) _inputBuffer[_inputPtr++];
        if ((c & 0xC0) != 0x080) {
```

```
            _reportInvalidOther(c & 0xFF, _inputPtr);
        }
    }

    /* Alas, can't heavily optimize skipping, since we still have to
     * do validity checks...
     */
    private final void _skipUtf8_3(int c)
        throws IOException, JsonParseException
    {
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        //c &= 0x0F;
        c = (int) _inputBuffer[_inputPtr++];
        if ((c & 0xC0) != 0x080) {
            _reportInvalidOther(c & 0xFF, _inputPtr);
        }
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        c = (int) _inputBuffer[_inputPtr++];
        if ((c & 0xC0) != 0x080) {
            _reportInvalidOther(c & 0xFF, _inputPtr);
        }
    }

    private final void _skipUtf8_4(int c)
        throws IOException, JsonParseException
    {
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        int d = (int) _inputBuffer[_inputPtr++];
        if ((d & 0xC0) != 0x080) {
            _reportInvalidOther(d & 0xFF, _inputPtr);
        }
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        if ((d & 0xC0) != 0x080) {
            _reportInvalidOther(d & 0xFF, _inputPtr);
        }
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        d = (int) _inputBuffer[_inputPtr++];
        if ((d & 0xC0) != 0x080) {
```

```
            _reportInvalidOther(d & 0xFF, _inputPtr);
        }
    }

    /*
    /**********************************************************
    /* Internal methods, input loading
    /**********************************************************
     */

    /**
     * We actually need to check the character value here
     * (to see if we have \n following \r).
     */
    protected final void _skipCR() throws IOException
    {
        if (_inputPtr < _inputEnd || loadMore()) {
            if (_inputBuffer[_inputPtr] == BYTE_LF) {
                ++_inputPtr;
            }
        }
        ++_currInputRow;
        _currInputRowStart = _inputPtr;
    }

    protected final void _skipLF() throws IOException
    {
        ++_currInputRow;
        _currInputRowStart = _inputPtr;
    }

    private int nextByte()
        throws IOException, JsonParseException
    {
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        return _inputBuffer[_inputPtr++] & 0xFF;
    }

    /*
    /**********************************************************
    /* Internal methods, error reporting
    /**********************************************************
     */

    protected void _reportInvalidChar(int c)
        throws JsonParseException
```

```
    {
        // Either invalid WS or illegal UTF-8 start char
        if (c < INT_SPACE) {
            _throwInvalidSpace(c);
        }
        _reportInvalidInitial(c);
    }

protected void _reportInvalidInitial(int mask)
    throws JsonParseException
{
    _reportError("Invalid UTF-8 start byte 0x"+Integer.toHexString(mask));
}

protected void _reportInvalidOther(int mask)
    throws JsonParseException
{
    _reportError("Invalid UTF-8 middle byte 0x"+Integer.toHexString(mask));
}

protected void _reportInvalidOther(int mask, int ptr)
    throws JsonParseException
{
    _inputPtr = ptr;
    _reportInvalidOther(mask);
}

public static int[] growArrayBy(int[] arr, int more)
{
    if (arr == null) {
        return new int[more];
    }
    int[] old = arr;
    int len = arr.length;
    arr = new int[len + more];
    System.arraycopy(old, 0, arr, 0, len);
    return arr;
}

/*
/**********************************************************
/* Binary access
/**********************************************************
 */

@Override
protected byte[] _decodeBase64(Base64Variant b64variant)
    throws IOException, JsonParseException
```

```
{
    ByteArrayBuilder builder = _getByteArrayBuilder();

    /* !!! 23-Jan-2009, tatu: There are some potential problems
     *   with this:
     *
     * - Escaped chars are not handled. Should they?
     */

    //main_loop:
    while (true) {
        // first, we'll skip preceding white space, if any
        int ch;
        do {
            if (_inputPtr >= _inputEnd) {
                loadMoreGuaranteed();
            }
            ch = (int) _inputBuffer[_inputPtr++] & 0xFF;
        } while (ch <= INT_SPACE);
        int bits = b64variant.decodeBase64Char(ch);
        if (bits < 0) { // reached the end, fair and square?
            if (ch == INT_QUOTE) {
                return builder.toByteArray();
            }
            throw reportInvalidChar(b64variant, ch, 0);
        }
        int decodedData = bits;

        // then second base64 char; can't get padding yet, nor ws

        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        ch = _inputBuffer[_inputPtr++] & 0xFF;
        bits = b64variant.decodeBase64Char(ch);
        if (bits < 0) {
            throw reportInvalidChar(b64variant, ch, 1);
        }
        decodedData = (decodedData << 6) | bits;

        // third base64 char; can be padding, but not ws
        if (_inputPtr >= _inputEnd) {
            loadMoreGuaranteed();
        }
        ch = _inputBuffer[_inputPtr++] & 0xFF;
        bits = b64variant.decodeBase64Char(ch);

        // First branch: can get padding (-> 1 byte)
```

```
            if (bits < 0) {
                if (bits != Base64Variant.BASE64_VALUE_PADDING) {
                    throw reportInvalidChar(b64variant, ch, 2);
                }
                // Ok, must get padding
                if (_inputPtr >= _inputEnd) {
                    loadMoreGuaranteed();
                }
                ch = _inputBuffer[_inputPtr++] & 0xFF;
                if (!b64variant.usesPaddingChar(ch)) {
                    throw reportInvalidChar(b64variant, ch, 3, "expected padding character
'"+b64variant.getPaddingChar()+"'");
                }
                // Got 12 bits, only need 8, need to shift
                decodedData >>= 4;
                builder.append(decodedData);
                continue;
            }
            // Nope, 2 or 3 bytes
            decodedData = (decodedData << 6) | bits;
            // fourth and last base64 char; can be padding, but not ws
            if (_inputPtr >= _inputEnd) {
                loadMoreGuaranteed();
            }
            ch = _inputBuffer[_inputPtr++] & 0xFF;
            bits = b64variant.decodeBase64Char(ch);
            if (bits < 0) {
                if (bits != Base64Variant.BASE64_VALUE_PADDING) {
                    throw reportInvalidChar(b64variant, ch, 3);
                }
                /* With padding we only get 2 bytes; but we have
                 * to shift it a bit so it is identical to triplet
                 * case with partial output.
                 * 3 chars gives 3x6 == 18 bits, of which 2 are
                 * dummies, need to discard:
                 */
                decodedData >>= 2;
                builder.appendTwoBytes(decodedData);
            } else {
                // otherwise, our triple is now complete
                decodedData = (decodedData << 6) | bits;
                builder.appendThreeBytes(decodedData);
            }
        }
    }

    protected IllegalArgumentException reportInvalidChar(Base64Variant b64variant, int ch, int bindex)
        throws IllegalArgumentException
```

```
  {
    return reportInvalidChar(b64variant, ch, bindex, null);
  }

  /**
   * @param bindex Relative index within base64 character unit; between 0
   *   and 3 (as unit has exactly 4 characters)
   */
  protected IllegalArgumentException reportInvalidChar(Base64Variant b64variant, int ch, int bindex, String msg)
    throws IllegalArgumentException
  {
    String base;
    if (ch <= INT_SPACE) {
      base = "Illegal white space character (code 0x"+Integer.toHexString(ch)+") as character #"+(bindex+1)+" of
4-char base64 unit: can only used between units";
    } else if (b64variant.usesPaddingChar(ch)) {
      base = "Unexpected padding character ('"+b64variant.getPaddingChar()+"') as character #"+(bindex+1)+" of
4-char base64 unit: padding only legal as 3rd or 4th character";
    } else if (!Character.isDefined(ch) || Character.isISOControl(ch)) {
      // Not sure if we can really get here... ? (most illegal xml chars are caught at lower level)
      base = "Illegal character (code 0x"+Integer.toHexString(ch)+") in base64 content";
    } else {
      base = "Illegal character '"+((char)ch)+"' (code 0x"+Integer.toHexString(ch)+") in base64 content";
    }
    if (msg != null) {
      base = base + ": " + msg;
    }
    return new IllegalArgumentException(base);
  }
}
package org.codehaus.jackson.impl;

import java.io.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.IOContext;
import org.codehaus.jackson.sym.CharsToNameCanonicalizer;
import org.codehaus.jackson.util.*;

/**
* This is a concrete implementation of {@link JsonParser}, which is
* based on a {@link java.io.Reader} to handle low-level character
* conversion tasks.
*/
public final class ReaderBasedParser
  extends ReaderBasedNumericParser
{
  /*
```

```
/**********************************************************
/* Configuration
/**********************************************************
 */

protected ObjectCodec _objectCodec;

final protected CharsToNameCanonicalizer _symbols;

/*
/**********************************************************
/* Parsing state
/**********************************************************
 */

/**
 * Flag that indicates that the current token has not yet
 * been fully processed, and needs to be finished for
 * some access (or skipped to obtain the next token)
 */
protected boolean _tokenIncomplete = false;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public ReaderBasedParser(IOContext ioCtxt, int features, Reader r,
                         ObjectCodec codec, CharsToNameCanonicalizer st)
{
   super(ioCtxt, features, r);
   _objectCodec = codec;
   _symbols = st;
}

@Override
public ObjectCodec getCodec() {
   return _objectCodec;
}

@Override
public void setCodec(ObjectCodec c) {
   _objectCodec = c;
}

/*
/**********************************************************
```

```
/* Public API, data access
/**********************************************************
 */

/**
 * Method for accessing textual representation of the current event;
 * if no current event (before first call to {@link #nextToken}, or
 * after encountering end-of-input), returns null.
 * Method can be called for any event.
 */
@Override
public final String getText()
   throws IOException, JsonParseException
{
   JsonToken t = _currToken;
   if (t == JsonToken.VALUE_STRING) {
      if (_tokenIncomplete) {
         _tokenIncomplete = false;
         _finishString(); // only strings can be incomplete
      }
      return _textBuffer.contentsAsString();
   }
   return _getText2(t);
}

protected final String _getText2(JsonToken t)
{
   if (t == null) {
      return null;
   }
   switch (t) {
   case FIELD_NAME:
      return _parsingContext.getCurrentName();

   case VALUE_STRING:
      // fall through
   case VALUE_NUMBER_INT:
   case VALUE_NUMBER_FLOAT:
      return _textBuffer.contentsAsString();
   }
   return t.asString();
}

@Override
public char[] getTextCharacters()
   throws IOException, JsonParseException
{
   if (_currToken != null) { // null only before/after document
```

```
        switch (_currToken) {

        case FIELD_NAME:
            if (!_nameCopied) {
                String name = _parsingContext.getCurrentName();
                int nameLen = name.length();
                if (_nameCopyBuffer == null) {
                    _nameCopyBuffer = _ioContext.allocNameCopyBuffer(nameLen);
                } else if (_nameCopyBuffer.length < nameLen) {
                    _nameCopyBuffer = new char[nameLen];
                }
                name.getChars(0, nameLen, _nameCopyBuffer, 0);
                _nameCopied = true;
            }
            return _nameCopyBuffer;

        case VALUE_STRING:
            if (_tokenIncomplete) {
                _tokenIncomplete = false;
                _finishString(); // only strings can be incomplete
            }
            // fall through
        case VALUE_NUMBER_INT:
        case VALUE_NUMBER_FLOAT:
            return _textBuffer.getTextBuffer();

        default:
            return _currToken.asCharArray();
        }
    }
    return null;
}

@Override
public int getTextLength()
    throws IOException, JsonParseException
{
    if (_currToken != null) { // null only before/after document
        switch (_currToken) {

        case FIELD_NAME:
            return _parsingContext.getCurrentName().length();
        case VALUE_STRING:
            if (_tokenIncomplete) {
                _tokenIncomplete = false;
                _finishString(); // only strings can be incomplete
            }
            // fall through
```

```
            case VALUE_NUMBER_INT:
            case VALUE_NUMBER_FLOAT:
                return _textBuffer.size();

            default:
                return _currToken.asCharArray().length;
            }
        }
        return 0;
    }

    @Override
    public int getTextOffset() throws IOException, JsonParseException
    {
        // Most have offset of 0, only some may have other values:
        if (_currToken != null) {
            switch (_currToken) {
            case FIELD_NAME:
                return 0;
            case VALUE_STRING:
                if (_tokenIncomplete) {
                    _tokenIncomplete = false;
                    _finishString(); // only strings can be incomplete
                }
                // fall through
            case VALUE_NUMBER_INT:
            case VALUE_NUMBER_FLOAT:
                return _textBuffer.getTextOffset();
            }
        }
        return 0;
    }

    @Override
    public byte[] getBinaryValue(Base64Variant b64variant)
        throws IOException, JsonParseException
    {
        if (_currToken != JsonToken.VALUE_STRING &&
                (_currToken != JsonToken.VALUE_EMBEDDED_OBJECT || _binaryValue == null)) {
            _reportError("Current token ("+_currToken+") not VALUE_STRING or VALUE_EMBEDDED_OBJECT,
can not access as binary");
        }
        /* To ensure that we won't see inconsistent data, better clear up
         * state...
         */
        if (_tokenIncomplete) {
            try {
                _binaryValue = _decodeBase64(b64variant);
```

```
            } catch (IllegalArgumentException iae) {
                throw _constructError("Failed to decode VALUE_STRING as base64 ("+b64variant+"):
"+iae.getMessage());
            }
            /* let's clear incomplete only now; allows for accessing other
             * textual content in error cases
             */
            _tokenIncomplete = false;
        }
        return _binaryValue;
    }

    /*
    /**********************************************************
    /* Public API, traversal
    /**********************************************************
     */

    /**
     * @return Next token from the stream, if any found, or null
     *   to indicate end-of-input
     */
    @Override
    public JsonToken nextToken()
        throws IOException, JsonParseException
    {
        /* First: field names are special -- we will always tokenize
         * (part of) value along with field name to simplify
         * state handling. If so, can and need to use secondary token:
         */
        if (_currToken == JsonToken.FIELD_NAME) {
            return _nextAfterName();
        }
        if (_tokenIncomplete) {
            _skipString(); // only strings can be partial
        }
        int i = _skipWSOrEnd();
        if (i < 0) { // end-of-input
            /* 19-Feb-2009, tatu: Should actually close/release things
             *    like input source, symbol table and recyclable buffers now.
             */
            close();
            return (_currToken = null);
        }

        /* First, need to ensure we know the starting location of token
         * after skipping leading white space
         */
```

```
        _tokenInputTotal = _currInputProcessed + _inputPtr - 1;
        _tokenInputRow = _currInputRow;
        _tokenInputCol = _inputPtr - _currInputRowStart - 1;

        // finally: clear any data retained so far
        _binaryValue = null;

        // Closing scope?
        if (i == INT_RBRACKET) {
            if (!_parsingContext.inArray()) {
                _reportMismatchedEndMarker(i, '}');
            }
            _parsingContext = _parsingContext.getParent();
            return (_currToken = JsonToken.END_ARRAY);
        }
        if (i == INT_RCURLY) {
            if (!_parsingContext.inObject()) {
                _reportMismatchedEndMarker(i, ']');
            }
            _parsingContext = _parsingContext.getParent();
            return (_currToken = JsonToken.END_OBJECT);
        }

        // Nope: do we then expect a comma?
        if (_parsingContext.expectComma()) {
            if (i != INT_COMMA) {
                _reportUnexpectedChar(i, "was expecting comma to separate "+_parsingContext.getTypeDesc()+"
entries");
            }
            i = _skipWS();
        }

        /* And should we now have a name? Always true for
         * Object contexts, since the intermediate 'expect-value'
         * state is never retained.
         */
        boolean inObject = _parsingContext.inObject();
        if (inObject) {
            // First, field name itself:
            String name = _parseFieldName(i);
            _parsingContext.setCurrentName(name);
            _currToken = JsonToken.FIELD_NAME;
            i = _skipWS();
            if (i != INT_COLON) {
                _reportUnexpectedChar(i, "was expecting a colon to separate field name and value");
            }
            i = _skipWS();
        }
```

```
// Ok: we must have a value... what is it?

JsonToken t;

switch (i) {
case INT_QUOTE:
   _tokenIncomplete = true;
   t = JsonToken.VALUE_STRING;
   break;
case INT_LBRACKET:
   if (!inObject) {
      _parsingContext = _parsingContext.createChildArrayContext(_tokenInputRow, _tokenInputCol);
   }
   t = JsonToken.START_ARRAY;
   break;
case INT_LCURLY:
   if (!inObject) {
      _parsingContext = _parsingContext.createChildObjectContext(_tokenInputRow, _tokenInputCol);
   }
   t = JsonToken.START_OBJECT;
   break;
case INT_RBRACKET:
case INT_RCURLY:
   // Error: neither is valid at this point; valid closers have
   // been handled earlier
   _reportUnexpectedChar(i, "expected a value");
case INT_t:
   _matchToken(JsonToken.VALUE_TRUE);
   t = JsonToken.VALUE_TRUE;
   break;
case INT_f:
   _matchToken(JsonToken.VALUE_FALSE);
   t = JsonToken.VALUE_FALSE;
   break;
case INT_n:
   _matchToken(JsonToken.VALUE_NULL);
   t = JsonToken.VALUE_NULL;
   break;

case INT_MINUS:
   /* Should we have separate handling for plus? Although
    * it is not allowed per se, it may be erroneously used,
    * and could be indicate by a more specific error message.
    */
case INT_0:
case INT_1:
case INT_2:
```

```
    case INT_3:
    case INT_4:
    case INT_5:
    case INT_6:
    case INT_7:
    case INT_8:
    case INT_9:
        t = parseNumberText(i);
        break;
    default:
        t = _handleUnexpectedValue(i);
        break;
    }

    if (inObject) {
        _nextToken = t;
        return _currToken;
    }
    _currToken = t;
    return t;
}

private final JsonToken _nextAfterName()
{
    _nameCopied = false; // need to invalidate if it was copied
    JsonToken t = _nextToken;
    _nextToken = null;
    // Also: may need to start new context?
    if (t == JsonToken.START_ARRAY) {
        _parsingContext = _parsingContext.createChildArrayContext(_tokenInputRow, _tokenInputCol);
    } else if (t == JsonToken.START_OBJECT) {
        _parsingContext = _parsingContext.createChildObjectContext(_tokenInputRow, _tokenInputCol);
    }
    return (_currToken = t);
}

@Override
public void close() throws IOException
{
    super.close();
    _symbols.release();
}

/*
/**********************************************************
/* Internal methods, secondary parsing
/**********************************************************
 */
```

```
protected final String _parseFieldName(int i)
    throws IOException, JsonParseException
{
    if (i != INT_QUOTE) {
        return _handleUnusualFieldName(i);
    }
    /* First: let's try to see if we have a simple name: one that does
     * not cross input buffer boundary, and does not contain escape
     * sequences.
     */
    int ptr = _inputPtr;
    int hash = 0;
    final int inputLen = _inputEnd;

    if (ptr < inputLen) {
        final int[] codes = CharTypes.getInputCodeLatin1();
        final int maxCode = codes.length;

        do {
            int ch = _inputBuffer[ptr];
            if (ch < maxCode && codes[ch] != 0) {
                if (ch == '"') {
                    int start = _inputPtr;
                    _inputPtr = ptr+1; // to skip the quote
                    return _symbols.findSymbol(_inputBuffer, start, ptr - start, hash);
                }
                break;
            }
            hash = (hash * 31) + ch;
            ++ptr;
        } while (ptr < inputLen);
    }

    int start = _inputPtr;
    _inputPtr = ptr;
    return _parseFieldName2(start, hash, INT_QUOTE);
}

private String _parseFieldName2(int startPtr, int hash, int endChar)
    throws IOException, JsonParseException
{
    _textBuffer.resetWithShared(_inputBuffer, startPtr, (_inputPtr - startPtr));

    /* Output pointers; calls will also ensure that the buffer is
     * not shared and has room for at least one more char.
     */
    char[] outBuf = _textBuffer.getCurrentSegment();
```

```
    int outPtr = _textBuffer.getCurrentSegmentSize();

    while (true) {
        if (_inputPtr >= _inputEnd) {
            if (!loadMore()) {
                _reportInvalidEOF(": was expecting closing '"+((char) endChar)+"' for name");
            }
        }
        char c = _inputBuffer[_inputPtr++];
        int i = (int) c;
        if (i <= INT_BACKSLASH) {
            if (i == INT_BACKSLASH) {
                /* Although chars outside of BMP are to be escaped as
                 * an UTF-16 surrogate pair, does that affect decoding?
                 * For now let's assume it does not.
                 */
                c = _decodeEscaped();
            } else if (i <= endChar) {
                if (i == endChar) {
                    break;
                }
                if (i < INT_SPACE) {
                    _throwUnquotedSpace(i, "name");
                }
            }
        }
        hash = (hash * 31) + i;
        // Ok, let's add char to output:
        outBuf[outPtr++] = c;

        // Need more room?
        if (outPtr >= outBuf.length) {
            outBuf = _textBuffer.finishCurrentSegment();
            outPtr = 0;
        }
    }
    _textBuffer.setCurrentLength(outPtr);
    {
        TextBuffer tb = _textBuffer;
        char[] buf = tb.getTextBuffer();
        int start = tb.getTextOffset();
        int len = tb.size();

        return _symbols.findSymbol(buf, start, len, hash);
    }
}

/**
```

```
     * Method called when we see non-white space character other
     * than double quote, when expecting a field name.
     * In standard mode will just throw an expection; but
     * in non-standard modes may be able to parse name.
     *
     * @since 1.2
     */
    protected final String _handleUnusualFieldName(int i)
        throws IOException, JsonParseException
    {
        // [JACKSON-173]: allow single quotes
        if (i == INT_APOSTROPHE && isEnabled(Feature.ALLOW_SINGLE_QUOTES)) {
            return _parseApostropheFieldName();
        }
        // [JACKSON-69]: allow unquoted names if feature enabled:
        if (!isEnabled(Feature.ALLOW_UNQUOTED_FIELD_NAMES)) {
            _reportUnexpectedChar(i, "was expecting double-quote to start field name");
        }
        final int[] codes = CharTypes.getInputCodeLatin1JsNames();
        final int maxCode = codes.length;

        // Also: first char must be a valid name char, but NOT be number
        boolean firstOk;

        if (i < maxCode) { // identifier, and not a number
            firstOk = (codes[i] == 0) && (i < INT_0 || i > INT_9);
        } else {
            firstOk = Character.isJavaIdentifierPart((char) i);
        }
        if (!firstOk) {
            _reportUnexpectedChar(i, "was expecting either valid name character (for unquoted name) or double-quote
(for quoted) to start field name");
        }
        int ptr = _inputPtr;
        int hash = 0;
        final int inputLen = _inputEnd;

        if (ptr < inputLen) {
            do {
                int ch = _inputBuffer[ptr];
                if (ch < maxCode) {
                    if (codes[ch] != 0) {
                        int start = _inputPtr-1; // -1 to bring back first char
                        _inputPtr = ptr;
                        return _symbols.findSymbol(_inputBuffer, start, ptr - start, hash);
                    }
                } else if (!Character.isJavaIdentifierPart((char) ch)) {
                    int start = _inputPtr-1; // -1 to bring back first char
```

```
                _inputPtr = ptr;
                return _symbols.findSymbol(_inputBuffer, start, ptr - start, hash);
            }
            hash = (hash * 31) + ch;
            ++ptr;
        } while (ptr < inputLen);
    }
    int start = _inputPtr-1;
    _inputPtr = ptr;
    return _parseUnusualFieldName2(start, hash, codes);
}

protected final String _parseApostropheFieldName()
    throws IOException, JsonParseException
{
    // Note: mostly copy of_parseFieldName
    int ptr = _inputPtr;
    int hash = 0;
    final int inputLen = _inputEnd;

    if (ptr < inputLen) {
        final int[] codes = CharTypes.getInputCodeLatin1();
        final int maxCode = codes.length;

        do {
            int ch = _inputBuffer[ptr];
            if (ch == '\'') {
                int start = _inputPtr;
                _inputPtr = ptr+1; // to skip the quote
                return _symbols.findSymbol(_inputBuffer, start, ptr - start, hash);
            }
            if (ch < maxCode && codes[ch] != 0) {
                break;
            }
            hash = (hash * 31) + ch;
            ++ptr;
        } while (ptr < inputLen);
    }

    int start = _inputPtr;
    _inputPtr = ptr;

    return _parseFieldName2(start, hash, INT_APOSTROPHE);
}

/**
 * Method for handling cases where first non-space character
 * of an expected value token is not legal for standard JSON content.
```

```
     *
     * @since 1.3
     */
    protected final JsonToken _handleUnexpectedValue(int i)
        throws IOException, JsonParseException
    {
        // Most likely an error, unless we are to allow single-quote-strings
        if (i != INT_APOSTROPHE || !isEnabled(Feature.ALLOW_SINGLE_QUOTES)) {
            _reportUnexpectedChar(i, "expected a valid value (number, String, array, object, 'true', 'false' or 'null')");
        }

        /* [JACKSON-173]: allow single quotes. Unlike with regular
         * Strings, we'll eagerly parse contents; this so that there's
         * no need to store information on quote char used.
         *
         * Also, no separation to fast/slow parsing; we'll just do
         * one regular (~= slow) parsing, to keep code simple
         */
        char[] outBuf = _textBuffer.emptyAndGetCurrentSegment();
        int outPtr = _textBuffer.getCurrentSegmentSize();

        while (true) {
            if (_inputPtr >= _inputEnd) {
                if (!loadMore()) {
                    _reportInvalidEOF(": was expecting closing quote for a string value");
                }
            }
            char c = _inputBuffer[_inputPtr++];
            i = (int) c;
            if (i <= INT_BACKSLASH) {
                if (i == INT_BACKSLASH) {
                    /* Although chars outside of BMP are to be escaped as
                     * an UTF-16 surrogate pair, does that affect decoding?
                     * For now let's assume it does not.
                     */
                    c = _decodeEscaped();
                } else if (i <= INT_APOSTROPHE) {
                    if (i == INT_APOSTROPHE) {
                        break;
                    }
                    if (i < INT_SPACE) {
                        _throwUnquotedSpace(i, "string value");
                    }
                }
            }
            // Need more room?
            if (outPtr >= outBuf.length) {
                outBuf = _textBuffer.finishCurrentSegment();
```

```
            outPtr = 0;
        }
        // Ok, let's add char to output:
        outBuf[outPtr++] = c;
    }
    _textBuffer.setCurrentLength(outPtr);
    return JsonToken.VALUE_STRING;
}

/**
 * @since 1.2
 */
private String _parseUnusualFieldName2(int startPtr, int hash, int[] codes)
    throws IOException, JsonParseException
{
    _textBuffer.resetWithShared(_inputBuffer, startPtr, (_inputPtr - startPtr));
    char[] outBuf = _textBuffer.getCurrentSegment();
    int outPtr = _textBuffer.getCurrentSegmentSize();
    final int maxCode = codes.length;

    while (true) {
        if (_inputPtr >= _inputEnd) {
            if (!loadMore()) { // acceptable for now (will error out later)
                break;
            }
        }
        char c = _inputBuffer[_inputPtr];
        int i = (int) c;
        if (i <= maxCode) {
            if (codes[i] != 0) {
                break;
            }
        } else if (!Character.isJavaIdentifierPart(c)) {
            break;
        }
        ++_inputPtr;
        hash = (hash * 31) + i;
        // Ok, let's add char to output:
        outBuf[outPtr++] = c;

        // Need more room?
        if (outPtr >= outBuf.length) {
            outBuf = _textBuffer.finishCurrentSegment();
            outPtr = 0;
        }
    }
    _textBuffer.setCurrentLength(outPtr);
    {
```

```
        TextBuffer tb = _textBuffer;
        char[] buf = tb.getTextBuffer();
        int start = tb.getTextOffset();
        int len = tb.size();

        return _symbols.findSymbol(buf, start, len, hash);
    }
}

@Override
protected void _finishString()
    throws IOException, JsonParseException
{
    /* First: let's try to see if we have simple String value: one
     * that does not cross input buffer boundary, and does not
     * contain escape sequences.
     */
    int ptr = _inputPtr;
    final int inputLen = _inputEnd;

    if (ptr < inputLen) {
        final int[] codes = CharTypes.getInputCodeLatin1();
        final int maxCode = codes.length;

        do {
            int ch = _inputBuffer[ptr];
            if (ch < maxCode && codes[ch] != 0) {
                if (ch == "") {
                    _textBuffer.resetWithShared(_inputBuffer, _inputPtr, (ptr-_inputPtr));
                    _inputPtr = ptr+1;
                    // Yes, we got it all
                    return;
                }
                break;
            }
            ++ptr;
        } while (ptr < inputLen);
    }

    /* Either ran out of input, or bumped into an escape
     * sequence...
     */
    _textBuffer.resetWithCopy(_inputBuffer, _inputPtr, (ptr-_inputPtr));
    _inputPtr = ptr;
    _finishString2();
}

protected void _finishString2()
```

```
    throws IOException, JsonParseException
{
    char[] outBuf = _textBuffer.getCurrentSegment();
    int outPtr = _textBuffer.getCurrentSegmentSize();

    while (true) {
        if (_inputPtr >= _inputEnd) {
            if (!loadMore()) {
                _reportInvalidEOF(": was expecting closing quote for a string value");
            }
        }
        char c = _inputBuffer[_inputPtr++];
        int i = (int) c;
        if (i <= INT_BACKSLASH) {
            if (i == INT_BACKSLASH) {
                /* Although chars outside of BMP are to be escaped as
                 * an UTF-16 surrogate pair, does that affect decoding?
                 * For now let's assume it does not.
                 */
                c = _decodeEscaped();
            } else if (i <= INT_QUOTE) {
                if (i == INT_QUOTE) {
                    break;
                }
                if (i < INT_SPACE) {
                    _throwUnquotedSpace(i, "string value");
                }
            }
        }
        // Need more room?
        if (outPtr >= outBuf.length) {
            outBuf = _textBuffer.finishCurrentSegment();
            outPtr = 0;
        }
        // Ok, let's add char to output:
        outBuf[outPtr++] = c;
    }
    _textBuffer.setCurrentLength(outPtr);
}

/**
 * Method called to skim through rest of unparsed String value,
 * if it is not needed. This can be done bit faster if contents
 * need not be stored for future access.
 */
protected void _skipString()
    throws IOException, JsonParseException
{
```

```
    _tokenIncomplete = false;

    int inputPtr = _inputPtr;
    int inputLen = _inputEnd;
    char[] inputBuffer = _inputBuffer;

    while (true) {
        if (inputPtr >= inputLen) {
            _inputPtr = inputPtr;
            if (!loadMore()) {
                _reportInvalidEOF(": was expecting closing quote for a string value");
            }
            inputPtr = _inputPtr;
            inputLen = _inputEnd;
        }
        char c = inputBuffer[inputPtr++];
        int i = (int) c;
        if (i <= INT_BACKSLASH) {
            if (i == INT_BACKSLASH) {
                /* Although chars outside of BMP are to be escaped as
                 * an UTF-16 surrogate pair, does that affect decoding?
                 * For now let's assume it does not.
                 */
                _inputPtr = inputPtr;
                c = _decodeEscaped();
                inputPtr = _inputPtr;
                inputLen = _inputEnd;
            } else if (i <= INT_QUOTE) {
                if (i == INT_QUOTE) {
                    _inputPtr = inputPtr;
                    break;
                }
                if (i < INT_SPACE) {
                    _inputPtr = inputPtr;
                    _throwUnquotedSpace(i, "string value");
                }
            }
        }
    }
}

/**
 * Method called to much one of literal tokens we may expect
 */
protected void _matchToken(JsonToken token)
    throws IOException, JsonParseException
{
    // First char is already matched, need to check the rest
```

```
    String matchStr = token.asString();
    int i = 1;

    for (int len = matchStr.length(); i < len; ++i) {
        if (_inputPtr >= _inputEnd) {
            if (!loadMore()) {
                _reportInvalidEOF(" in a value");
            }
        }
        char c = _inputBuffer[_inputPtr];
        if (c != matchStr.charAt(i)) {
            _reportInvalidToken(matchStr.substring(0, i));
        }
        ++_inputPtr;
    }
    /* Ok, fine; let's not bother checking anything beyond keyword.
     * If there's something wrong there, it'll cause a parsing
     * error later on.
     */
    return;
}

private void _reportInvalidToken(String matchedPart)
    throws IOException, JsonParseException
{
    StringBuilder sb = new StringBuilder(matchedPart);
    /* Let's just try to find what appears to be the token, using
     * regular Java identifier character rules. It's just a heuristic,
     * nothing fancy here.
     */
    while (true) {
        if (_inputPtr >= _inputEnd) {
            if (!loadMore()) {
                break;
            }
        }
        char c = _inputBuffer[_inputPtr];
        if (!Character.isJavaIdentifierPart(c)) {
            break;
        }
        ++_inputPtr;
        sb.append(c);
    }

    _reportError("Unrecognized token '"+sb.toString()+"': was expecting 'null', 'true' or 'false'");
}

/*
```

```
/**********************************************************
/* Internal methods, other parsing
/**********************************************************
 */

/**
 * We actually need to check the character value here
 * (to see if we have \n following \r).
 */
protected final void _skipCR() throws IOException
{
    if (_inputPtr < _inputEnd || loadMore()) {
        if (_inputBuffer[_inputPtr] == '\n') {
            ++_inputPtr;
        }
    }
    ++_currInputRow;
    _currInputRowStart = _inputPtr;
}

protected final void _skipLF() throws IOException
{
    ++_currInputRow;
    _currInputRowStart = _inputPtr;
}

private final int _skipWS()
    throws IOException, JsonParseException
{
    while (_inputPtr < _inputEnd || loadMore()) {
        int i = (int) _inputBuffer[_inputPtr++];
        if (i > INT_SPACE) {
            if (i != INT_SLASH) {
                return i;
            }
            _skipComment();
        } else if (i != INT_SPACE) {
            if (i == INT_LF) {
                _skipLF();
            } else if (i == INT_CR) {
                _skipCR();
            } else if (i != INT_TAB) {
                _throwInvalidSpace(i);
            }
        }
    }
    throw _constructError("Unexpected end-of-input within/between "+_parsingContext.getTypeDesc()+" entries");
}
```

```
  private final int _skipWSOrEnd()
     throws IOException, JsonParseException
  {
     while ((_inputPtr < _inputEnd) || loadMore()) {
        int i = (int) _inputBuffer[_inputPtr++];
        if (i > INT_SPACE) {
           if (i != INT_SLASH) {
              return i;
           }
           _skipComment();
        } else if (i != INT_SPACE) {
           if (i == INT_LF) {
              _skipLF();
           } else if (i == INT_CR) {
              _skipCR();
           } else if (i != INT_TAB) {
              _throwInvalidSpace(i);
           }
        }
     }
     // We ran out of input...
     _handleEOF();
     return -1;
  }

  private final void _skipComment()
     throws IOException, JsonParseException
  {
     if (!isEnabled(Feature.ALLOW_COMMENTS)) {
        _reportUnexpectedChar('/', "maybe a (non-standard) comment? (not recognized as one since Feature
'ALLOW_COMMENTS' not enabled for parser)");
     }
     // First: check which comment (if either) it is:
     if (_inputPtr >= _inputEnd && !loadMore()) {
        _reportInvalidEOF(" in a comment");
     }
     char c = _inputBuffer[_inputPtr++];
     if (c == '/') {
        _skipCppComment();
     } else if (c == '*') {
        _skipCComment();
     } else {
        _reportUnexpectedChar(c, "was expecting either '*' or '/' for a comment");
     }
  }

  private final void _skipCComment()
```

```
    throws IOException, JsonParseException
{
    // Ok: need the matching '*/'
    main_loop:
    while ((_inputPtr < _inputEnd) || loadMore()) {
        int i = (int) _inputBuffer[_inputPtr++];
        if (i <= INT_ASTERISK) {
            if (i == INT_ASTERISK) { // end?
                if ((_inputPtr >= _inputEnd) && !loadMore()) {
                    break main_loop;
                }
                if (_inputBuffer[_inputPtr] == INT_SLASH) {
                    ++_inputPtr;
                    return;
                }
                continue;
            }
            if (i < INT_SPACE) {
                if (i == INT_LF) {
                    _skipLF();
                } else if (i == INT_CR) {
                    _skipCR();
                } else if (i != INT_TAB) {
                    _throwInvalidSpace(i);
                }
            }
        }
    }
    _reportInvalidEOF(" in a comment");
}

private final void _skipCppComment()
    throws IOException, JsonParseException
{
    // Ok: need to find EOF or linefeed
    while ((_inputPtr < _inputEnd) || loadMore()) {
        int i = (int) _inputBuffer[_inputPtr++];
        if (i < INT_SPACE) {
            if (i == INT_LF) {
                _skipLF();
                break;
            } else if (i == INT_CR) {
                _skipCR();
                break;
            } else if (i != INT_TAB) {
                _throwInvalidSpace(i);
            }
        }
```

```
    }
}

protected final char _decodeEscaped()
    throws IOException, JsonParseException
{
    if (_inputPtr >= _inputEnd) {
        if (!loadMore()) {
            _reportInvalidEOF(" in character escape sequence");
        }
    }
    char c = _inputBuffer[_inputPtr++];

    switch ((int) c) {
        // First, ones that are mapped
    case INT_b:
        return '\b';
    case INT_t:
        return '\t';
    case INT_n:
        return '\n';
    case INT_f:
        return '\f';
    case INT_r:
        return '\r';

        // And these are to be returned as they are
    case INT_QUOTE:
    case INT_SLASH:
    case INT_BACKSLASH:
        return c;

    case INT_u: // and finally hex-escaped
        break;

    default:
        return _handleUnrecognizedCharacterEscape(c);
    }

    // Ok, a hex escape. Need 4 characters
    int value = 0;
    for (int i = 0; i < 4; ++i) {
        if (_inputPtr >= _inputEnd) {
            if (!loadMore()) {
                _reportInvalidEOF(" in character escape sequence");
            }
        }
        int ch = (int) _inputBuffer[_inputPtr++];
```

```
        int digit = CharTypes.charToHex(ch);
        if (digit < 0) {
            _reportUnexpectedChar(ch, "expected a hex-digit for character escape sequence");
        }
        value = (value << 4) | digit;
    }
    return (char) value;
}

/*
/**********************************************************
/* Binary access
/**********************************************************
 */

@Override
protected byte[] _decodeBase64(Base64Variant b64variant)
    throws IOException, JsonParseException
{
    ByteArrayBuilder builder = _getByteArrayBuilder();

    /* !!! 23-Jan-2009, tatu: There are some potential problems
     *   with this:
     *
     * - Escaped chars are not handled. Should they?
     */

    //main_loop:
    while (true) {
        // first, we'll skip preceding white space, if any
        char ch;
        do {
            if (_inputPtr >= _inputEnd) {
                loadMoreGuaranteed();
            }
            ch = _inputBuffer[_inputPtr++];
        } while (ch <= INT_SPACE);
        int bits = b64variant.decodeBase64Char(ch);
        if (bits < 0) { // reached the end, fair and square?
            if (ch == '"') {
                return builder.toByteArray();
            }
            throw reportInvalidChar(b64variant, ch, 0);
        }
        int decodedData = bits;

        // then second base64 char; can't get padding yet, nor ws
```

```
            if (_inputPtr >= _inputEnd) {
               loadMoreGuaranteed();
            }
            ch = _inputBuffer[_inputPtr++];
            bits = b64variant.decodeBase64Char(ch);
            if (bits < 0) {
               throw reportInvalidChar(b64variant, ch, 1);
            }
            decodedData = (decodedData << 6) | bits;

            // third base64 char; can be padding, but not ws
            if (_inputPtr >= _inputEnd) {
               loadMoreGuaranteed();
            }
            ch = _inputBuffer[_inputPtr++];
            bits = b64variant.decodeBase64Char(ch);

            // First branch: can get padding (-> 1 byte)
            if (bits < 0) {
               if (bits != Base64Variant.BASE64_VALUE_PADDING) {
                  throw reportInvalidChar(b64variant, ch, 2);
               }
               // Ok, must get padding
               if (_inputPtr >= _inputEnd) {
                  loadMoreGuaranteed();
               }
               ch = _inputBuffer[_inputPtr++];
               if (!b64variant.usesPaddingChar(ch)) {
                  throw reportInvalidChar(b64variant, ch, 3, "expected padding character
'"+b64variant.getPaddingChar()+"'");
               }
               // Got 12 bits, only need 8, need to shift
               decodedData >>= 4;
               builder.append(decodedData);
               continue;
            }
            // Nope, 2 or 3 bytes
            decodedData = (decodedData << 6) | bits;
            // fourth and last base64 char; can be padding, but not ws
            if (_inputPtr >= _inputEnd) {
               loadMoreGuaranteed();
            }
            ch = _inputBuffer[_inputPtr++];
            bits = b64variant.decodeBase64Char(ch);
            if (bits < 0) {
               if (bits != Base64Variant.BASE64_VALUE_PADDING) {
                  throw reportInvalidChar(b64variant, ch, 3);
               }
```

```
            /* With padding we only get 2 bytes; but we have
             * to shift it a bit so it is identical to triplet
             * case with partial output.
             * 3 chars gives 3x6 == 18 bits, of which 2 are
             * dummies, need to discard:
             */
            decodedData >>= 2;
            builder.appendTwoBytes(decodedData);
          } else {
            // otherwise, our triple is now complete
            decodedData = (decodedData << 6) | bits;
            builder.appendThreeBytes(decodedData);
          }
        }
      }

      protected IllegalArgumentException reportInvalidChar(Base64Variant b64variant, char ch, int bindex)
        throws IllegalArgumentException
      {
        return reportInvalidChar(b64variant, ch, bindex, null);
      }

      /**
       * @param bindex Relative index within base64 character unit; between 0
       *   and 3 (as unit has exactly 4 characters)
       */
      protected IllegalArgumentException reportInvalidChar(Base64Variant b64variant, char ch, int bindex, String msg)
        throws IllegalArgumentException
      {
        String base;
        if (ch <= INT_SPACE) {
          base = "Illegal white space character (code 0x"+Integer.toHexString(ch)+") as character #"+(bindex+1)+" of
    4-char base64 unit: can only used between units";
        } else if (b64variant.usesPaddingChar(ch)) {
          base = "Unexpected padding character ('"+b64variant.getPaddingChar()+"') as character #"+(bindex+1)+" of
    4-char base64 unit: padding only legal as 3rd or 4th character";
        } else if (!Character.isDefined(ch) || Character.isISOControl(ch)) {
          // Not sure if we can really get here... ? (most illegal xml chars are caught at lower level)
          base = "Illegal character (code 0x"+Integer.toHexString(ch)+") in base64 content";
        } else {
          base = "Illegal character '"+ch+"' (code 0x"+Integer.toHexString(ch)+") in base64 content";
        }
        if (msg != null) {
          base = base + ": " + msg;
        }
        return new IllegalArgumentException(base);
      }
    }
```

```
package org.codehaus.jackson.impl;

import java.io.*;

import org.codehaus.jackson.io.IOContext;

/**
* This is a simple low-level input reader base class, used by
* JSON parser. It is used when underlying input source is
* a byte stream such as {@link InputStream}.
* The reason for sub-classing (over composition)
* is due to need for direct access to low-level byte buffers
* and positions.
*
* @author Tatu Saloranta
*/
public abstract class StreamBasedParserBase
  extends JsonNumericParserBase
{
  /*
  /**********************************************************
  /* Configuration
  /**********************************************************
   */

  /**
   * Input stream that can be used for reading more content, if one
   * in use. May be null, if input comes just as a full buffer,
   * or if the stream has been closed.
   */
  protected InputStream _inputStream;

  /*
  /**********************************************************
  /* Current input data
  /**********************************************************
   */

  /**
   * Current buffer from which data is read; generally data is read into
   * buffer from input source, but in some cases pre-loaded buffer
   * is handed to the parser.
   */
  protected byte[] _inputBuffer;

  /**
   * Flag that indicates whether the input buffer is recycable (and
   * needs to be returned to recycler once we are done) or not.
```

```
 *<p>
 * If it is not, it also means that parser can NOT modify underlying
 * buffer.
 */
protected boolean _bufferRecyclable;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

protected StreamBasedParserBase(IOContext ctxt, int features,
                        InputStream in,
                        byte[] inputBuffer, int start, int end,
                        boolean bufferRecyclable)
{
   super(ctxt, features);
   _inputStream = in;
   _inputBuffer = inputBuffer;
   _inputPtr = start;
   _inputEnd = end;
   _bufferRecyclable = bufferRecyclable;
}

/*
/**********************************************************
/* Overrides
/**********************************************************
 */

@Override
public int releaseBuffered(OutputStream out) throws IOException
{
   int count = _inputEnd - _inputPtr;
   if (count < 1) {
      return 0;
   }
   // let's just advance ptr to end
   int origPtr = _inputPtr;
   out.write(_inputBuffer, origPtr, count);
   return count;
}

/*
/**********************************************************
/* Low-level reading, other
/**********************************************************
```

```
     */

    @Override
    protected final boolean loadMore()
        throws IOException
    {
        _currInputProcessed += _inputEnd;
        _currInputRowStart -= _inputEnd;

        if (_inputStream != null) {
            int count = _inputStream.read(_inputBuffer, 0, _inputBuffer.length);
            if (count > 0) {
                _inputPtr = 0;
                _inputEnd = count;
                return true;
            }
            // End of input
            _closeInput();
            // Should never return 0, so let's fail
            if (count == 0) {
                throw new IOException("InputStream.read() returned 0 characters when trying to read
"+_inputBuffer.length+" bytes");
            }
        }
        return false;
    }

    /**
     * Helper method that will try to load at least specified number bytes in
     * input buffer, possible moving existing data around if necessary
     *
     * @since 1.6
     */
    protected final boolean _loadToHaveAtLeast(int minAvailable)
        throws IOException
    {
        // No input stream, no leading (either we are closed, or have non-stream input source)
        if (_inputStream == null) {
            return false;
        }
        // Need to move remaining data in front?
        int amount = _inputEnd - _inputPtr;
        if (amount > 0 && _inputPtr > 0) {
            _currInputProcessed += _inputPtr;
            _currInputRowStart -= _inputPtr;
            System.arraycopy(_inputBuffer, _inputPtr, _inputBuffer, 0, amount);
            _inputEnd = amount;
        } else {
```

```
            _inputEnd = 0;
        }
        _inputPtr = 0;
        while (_inputEnd < minAvailable) {
            int count = _inputStream.read(_inputBuffer, _inputEnd, _inputBuffer.length - _inputEnd);
            if (count < 1) {
                // End of input
                _closeInput();
                // Should never return 0, so let's fail
                if (count == 0) {
                    throw new IOException("InputStream.read() returned 0 characters when trying to read "+amount+"
bytes");
                }
                return false;
            }
            _inputEnd += count;
        }
        return true;
    }

    @Override
    protected void _closeInput() throws IOException
    {
        /* 25-Nov-2008, tatus: As per [JACKSON-16] we are not to call close()
         *   on the underlying InputStream, unless we "own" it, or auto-closing
         *   feature is enabled.
         */
        if (_inputStream != null) {
            if (_ioContext.isResourceManaged() || isEnabled(Feature.AUTO_CLOSE_SOURCE)) {
                _inputStream.close();
            }
            _inputStream = null;
        }
    }

    /**
     * Method called to release internal buffers owned by the base
     * reader. This may be called along with {@link #_closeInput} (for
     * example, when explicitly closing this reader instance), or
     * separately (if need be).
     */
    @Override
    protected void _releaseBuffers() throws IOException
    {
        super._releaseBuffers();
        if (_bufferRecyclable) {
            byte[] buf = _inputBuffer;
            if (buf != null) {
```

```
            _inputBuffer = null;
            _ioContext.releaseReadIOBuffer(buf);
          }
        }
      }
    }
package org.codehaus.jackson.impl;

import java.io.*;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.*;
import org.codehaus.jackson.util.CharTypes;

/**
* {@link JsonGenerator} that outputs JSON content using a {@link java.io.Writer}
* which handles character encoding.
*/
public final class WriterBasedGenerator
  extends JsonGeneratorBase
{
  final protected static int SHORT_WRITE = 32;

  final protected static char[] HEX_CHARS = CharTypes.copyHexChars();

  /*
  /**********************************************************
  /* Configuration
  /**********************************************************
   */

  final protected IOContext _ioContext;

  final protected Writer _writer;

  /*
  /**********************************************************
  /* Output buffering
  /**********************************************************
   */

  /**
   * Intermediate buffer in which contents are buffered before
   * being written using {@link #_writer}.
   */
  protected char[] _outputBuffer;
```

```
/**
 * Pointer to the first buffered character to output
 */
protected int _outputHead = 0;

/**
 * Pointer to the position right beyond the last character to output
 * (end marker; may be past the buffer)
 */
protected int _outputTail = 0;

/**
 * End marker of the output buffer; one past the last valid position
 * within the buffer.
 */
protected int _outputEnd;

/**
 * 6-char temporary buffer allocated if needed, for constructing
 * escape sequences
 */
protected char[] _entityBuffer;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public WriterBasedGenerator(IOContext ctxt, int features, ObjectCodec codec,
                            Writer w)
{
   super(features, codec);
   _ioContext = ctxt;
   _writer = w;
   _outputBuffer = ctxt.allocConcatBuffer();
   _outputEnd = _outputBuffer.length;
}

/*
/**********************************************************
/* Overridden methods
/**********************************************************
 */

/* Most overrides in this section are just to make methods final,
 * to allow better inlining...
```

```
 */

@Override
public final void writeFieldName(String name)  throws IOException, JsonGenerationException
{
   int status = _writeContext.writeFieldName(name);
   if (status == JsonWriteContext.STATUS_EXPECT_VALUE) {
      _reportError("Can not write a field name, expecting a value");
   }
   _writeFieldName(name, (status == JsonWriteContext.STATUS_OK_AFTER_COMMA));
}

@Override
public final void writeStringField(String fieldName, String value)
   throws IOException, JsonGenerationException
{
   writeFieldName(fieldName);
   writeString(value);
}

@Override
public final void writeFieldName(SerializedString name)
   throws IOException, JsonGenerationException
{
   // Object is a value, need to verify it's allowed
   int status = _writeContext.writeFieldName(name.getValue());
   if (status == JsonWriteContext.STATUS_EXPECT_VALUE) {
      _reportError("Can not write a field name, expecting a value");
   }
   _writeFieldName(name, (status == JsonWriteContext.STATUS_OK_AFTER_COMMA));
}

@Override
public final void writeFieldName(SerializableString name)
   throws IOException, JsonGenerationException
{
   // Object is a value, need to verify it's allowed
   int status = _writeContext.writeFieldName(name.getValue());
   if (status == JsonWriteContext.STATUS_EXPECT_VALUE) {
      _reportError("Can not write a field name, expecting a value");
   }
   _writeFieldName(name, (status == JsonWriteContext.STATUS_OK_AFTER_COMMA));
}

/*
/**********************************************************
/* Output method implementations, structural
/**********************************************************
```

```
 */

@Override
public final void writeStartArray() throws IOException, JsonGenerationException
{
   _verifyValueWrite("start an array");
   _writeContext = _writeContext.createChildArrayContext();
   if (_cfgPrettyPrinter != null) {
      _cfgPrettyPrinter.writeStartArray(this);
   } else {
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail++] = '[';
   }
}

@Override
public final void writeEndArray() throws IOException, JsonGenerationException
{
   if (!_writeContext.inArray()) {
      _reportError("Current context not an ARRAY but "+_writeContext.getTypeDesc());
   }
   if (_cfgPrettyPrinter != null) {
      _cfgPrettyPrinter.writeEndArray(this, _writeContext.getEntryCount());
   } else {
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail++] = ']';
   }
   _writeContext = _writeContext.getParent();
}

@Override
public final void writeStartObject() throws IOException, JsonGenerationException
{
   _verifyValueWrite("start an object");
   _writeContext = _writeContext.createChildObjectContext();
   if (_cfgPrettyPrinter != null) {
      _cfgPrettyPrinter.writeStartObject(this);
   } else {
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail++] = '{';
   }
}
```

```
@Override
public final void writeEndObject() throws IOException, JsonGenerationException
{
   if (!_writeContext.inObject()) {
      _reportError("Current context not an object but "+_writeContext.getTypeDesc());
   }
   _writeContext = _writeContext.getParent();
   if (_cfgPrettyPrinter != null) {
      _cfgPrettyPrinter.writeEndObject(this, _writeContext.getEntryCount());
   } else {
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail++] = '}';
   }
}

protected void _writeFieldName(String name, boolean commaBefore)
   throws IOException, JsonGenerationException
{
   if (_cfgPrettyPrinter != null) {
      _writePPFieldName(name, commaBefore);
      return;
   }
   // for fast+std case, need to output up to 2 chars, comma, dquote
   if ((_outputTail + 1) >= _outputEnd) {
      _flushBuffer();
   }
   if (commaBefore) {
      _outputBuffer[_outputTail++] = ',';
   }

   /* To support [JACKSON-46], we'll do this:
    * (Quostion: should quoting of spaces (etc) still be enabled?)
    */
   if (!isEnabled(Feature.QUOTE_FIELD_NAMES)) {
      _writeString(name);
      return;
   }

   // we know there's room for at least one more char
   _outputBuffer[_outputTail++] = '"';
   // The beef:
   _writeString(name);
   // and closing quotes; need room for one more char:
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
```

```
    }
    _outputBuffer[_outputTail++] = '"';
}

public void _writeFieldName(SerializableString name, boolean commaBefore)
    throws IOException, JsonGenerationException
{
    if (_cfgPrettyPrinter != null) {
        _writePPFieldName(name, commaBefore);
        return;
    }
    // for fast+std case, need to output up to 2 chars, comma, dquote
    if ((_outputTail + 1) >= _outputEnd) {
        _flushBuffer();
    }
    if (commaBefore) {
        _outputBuffer[_outputTail++] = ',';
    }
    /* To support [JACKSON-46], we'll do this:
     * (Quostion: should quoting of spaces (etc) still be enabled?)
     */
    final char[] quoted = name.asQuotedChars();
    if (!isEnabled(Feature.QUOTE_FIELD_NAMES)) {
        writeRaw(quoted, 0, quoted.length);
        return;
    }
    // we know there's room for at least one more char
    _outputBuffer[_outputTail++] = '"';
    // The beef:
    final int qlen = quoted.length;
    if ((_outputTail + qlen + 1) >= _outputEnd) {
        writeRaw(quoted, 0, qlen);
        // and closing quotes; need room for one more char:
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = '"';
    } else {
        System.arraycopy(quoted, 0, _outputBuffer, _outputTail, qlen);
        _outputTail += qlen;
        _outputBuffer[_outputTail++] = '"';
    }
}

/**
 * Specialized version of <code>_writeFieldName</code>, off-lined
 * to keep the "fast path" as simple (and hopefully fast) as possible.
 */
```

```
protected final void _writePPFieldName(String name, boolean commaBefore)
   throws IOException, JsonGenerationException
{
   if (commaBefore) {
      _cfgPrettyPrinter.writeObjectEntrySeparator(this);
   } else {
      _cfgPrettyPrinter.beforeObjectEntries(this);
   }

   if (isEnabled(Feature.QUOTE_FIELD_NAMES)) { // standard
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail++] = '"';
      _writeString(name);
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail++] = '"';
   } else { // non-standard, omit quotes
      _writeString(name);
   }
}

protected final void _writePPFieldName(SerializableString name, boolean commaBefore)
   throws IOException, JsonGenerationException
{
   if (commaBefore) {
      _cfgPrettyPrinter.writeObjectEntrySeparator(this);
   } else {
      _cfgPrettyPrinter.beforeObjectEntries(this);
   }

   final char[] quoted = name.asQuotedChars();
   if (isEnabled(Feature.QUOTE_FIELD_NAMES)) { // standard
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail++] = '"';
      writeRaw(quoted, 0, quoted.length);
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail++] = '"';
   } else { // non-standard, omit quotes
      writeRaw(quoted, 0, quoted.length);
   }
}
```

```
/*
/**********************************************************
/* Output method implementations, textual
/**********************************************************
 */

@Override
public void writeString(String text)
   throws IOException, JsonGenerationException
{
   _verifyValueWrite("write text value");
   if (text == null) {
      _writeNull();
      return;
   }
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
   }
   _outputBuffer[_outputTail++] = '"';
   _writeString(text);
   // And finally, closing quotes
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
   }
   _outputBuffer[_outputTail++] = '"';
}

@Override
public void writeString(char[] text, int offset, int len)
   throws IOException, JsonGenerationException
{
   _verifyValueWrite("write text value");
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
   }
   _outputBuffer[_outputTail++] = '"';
   _writeString(text, offset, len);
   // And finally, closing quotes
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
   }
   _outputBuffer[_outputTail++] = '"';
}

@Override
public final void writeString(SerializableString sstr)
   throws IOException, JsonGenerationException
```

```
{
    _verifyValueWrite("write text value");
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = '"';
    // Note: copied from writeRaw:
    char[] text = sstr.asQuotedChars();
    final int len = text.length;
    // Only worth buffering if it's a short write?
    if (len < SHORT_WRITE) {
        int room = _outputEnd - _outputTail;
        if (len > room) {
            _flushBuffer();
        }
        System.arraycopy(text, 0, _outputBuffer, _outputTail, len);
        _outputTail += len;
    } else {
        // Otherwise, better just pass through:
        _flushBuffer();
        _writer.write(text, 0, len);
    }
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = '"';
}

@Override
public void writeRawUTF8String(byte[] text, int offset, int length)
    throws IOException, JsonGenerationException
{
    // could add support for buffering if we really want it...
    _reportUnsupportedOperation();
}

@Override
public void writeUTF8String(byte[] text, int offset, int length)
    throws IOException, JsonGenerationException
{
    // could add support for buffering if we really want it...
    _reportUnsupportedOperation();
}

/*
/**********************************************************
/* Output method implementations, unprocessed ("raw")
/**********************************************************
```

```
 */

@Override
public void writeRaw(String text)
   throws IOException, JsonGenerationException
{
   // Nothing to check, can just output as is
   int len = text.length();
   int room = _outputEnd - _outputTail;

   if (room == 0) {
      _flushBuffer();
      room = _outputEnd - _outputTail;
   }
   // But would it nicely fit in? If yes, it's easy
   if (room >= len) {
      text.getChars(0, len, _outputBuffer, _outputTail);
      _outputTail += len;
   } else {
      writeRawLong(text);
   }
}

@Override
public void writeRaw(String text, int start, int len)
   throws IOException, JsonGenerationException
{
   // Nothing to check, can just output as is
   int room = _outputEnd - _outputTail;

   if (room < len) {
      _flushBuffer();
      room = _outputEnd - _outputTail;
   }
   // But would it nicely fit in? If yes, it's easy
   if (room >= len) {
      text.getChars(start, start+len, _outputBuffer, _outputTail);
      _outputTail += len;
   } else {
      writeRawLong(text.substring(start, start+len));
   }
}

@Override
public void writeRaw(char[] text, int offset, int len)
   throws IOException, JsonGenerationException
{
   // Only worth buffering if it's a short write?
```

```
    if (len < SHORT_WRITE) {
        int room = _outputEnd - _outputTail;
        if (len > room) {
            _flushBuffer();
        }
        System.arraycopy(text, offset, _outputBuffer, _outputTail, len);
        _outputTail += len;
        return;
    }
    // Otherwise, better just pass through:
    _flushBuffer();
    _writer.write(text, offset, len);
}

@Override
public void writeRaw(char c)
    throws IOException, JsonGenerationException
{
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = c;
}

private void writeRawLong(String text)
    throws IOException, JsonGenerationException
{
    int room = _outputEnd - _outputTail;
    // If not, need to do it by looping
    text.getChars(0, room, _outputBuffer, _outputTail);
    _outputTail += room;
    _flushBuffer();
    int offset = room;
    int len = text.length() - room;

    while (len > _outputEnd) {
        int amount = _outputEnd;
        text.getChars(offset, offset+amount, _outputBuffer, 0);
        _outputHead = 0;
        _outputTail = amount;
        _flushBuffer();
        offset += amount;
        len -= amount;
    }
    // And last piece (at most length of buffer)
    text.getChars(offset, offset+len, _outputBuffer, 0);
    _outputHead = 0;
    _outputTail = len;
```

```
}

/*
/**********************************************************
/* Output method implementations, base64-encoded binary
/**********************************************************
 */

@Override
public void writeBinary(Base64Variant b64variant, byte[] data, int offset, int len)
   throws IOException, JsonGenerationException
{
   _verifyValueWrite("write binary value");
   // Starting quotes
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
   }
   _outputBuffer[_outputTail++] = '"';
   _writeBinary(b64variant, data, offset, offset+len);
   // and closing quotes
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
   }
   _outputBuffer[_outputTail++] = '"';
}

/*
/**********************************************************
/* Output method implementations, primitive
/**********************************************************
 */

@Override
public void writeNumber(int i)
   throws IOException, JsonGenerationException
{
   _verifyValueWrite("write number");
   // up to 10 digits and possible minus sign
   if ((_outputTail + 11) >= _outputEnd) {
      _flushBuffer();
   }
   if (_cfgNumbersAsStrings) {
      _writeQuotedInt(i);
      return;
   }
   _outputTail = NumberOutput.outputInt(i, _outputBuffer, _outputTail);
}
```

```
private final void _writeQuotedInt(int i) throws IOException {
    if ((_outputTail + 13) >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = '"';
    _outputTail = NumberOutput.outputInt(i, _outputBuffer, _outputTail);
    _outputBuffer[_outputTail++] = '"';
}

@Override
public void writeNumber(long l)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write number");
    if (_cfgNumbersAsStrings) {
        _writeQuotedLong(l);
        return;
    }
    if ((_outputTail + 21) >= _outputEnd) {
        // up to 20 digits, minus sign
        _flushBuffer();
    }
    _outputTail = NumberOutput.outputLong(l, _outputBuffer, _outputTail);
}

private final void _writeQuotedLong(long l) throws IOException {
    if ((_outputTail + 23) >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = '"';
    _outputTail = NumberOutput.outputLong(l, _outputBuffer, _outputTail);
    _outputBuffer[_outputTail++] = '"';
}

// !!! 05-Aug-2008, tatus: Any ways to optimize these?

@Override
public void writeNumber(BigInteger value)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write number");
    if (value == null) {
        _writeNull();
    } else if (_cfgNumbersAsStrings) {
        _writeQuotedRaw(value);
    } else {
        writeRaw(value.toString());
    }
```

```
}


@Override
public void writeNumber(double d)
   throws IOException, JsonGenerationException
{
   if (_cfgNumbersAsStrings ||
      // [JACKSON-139]
      (((Double.isNaN(d) || Double.isInfinite(d))
         && isEnabled(Feature.QUOTE_NON_NUMERIC_NUMBERS)))) {
      writeString(String.valueOf(d));
      return;
   }
   // What is the max length for doubles? 40 chars?
   _verifyValueWrite("write number");
   writeRaw(String.valueOf(d));
}

@Override
public void writeNumber(float f)
   throws IOException, JsonGenerationException
{
   if (_cfgNumbersAsStrings ||
      // [JACKSON-139]
      (((Float.isNaN(f) || Float.isInfinite(f))
         && isEnabled(Feature.QUOTE_NON_NUMERIC_NUMBERS)))) {
      writeString(String.valueOf(f));
      return;
   }
   // What is the max length for floats?
   _verifyValueWrite("write number");
   writeRaw(String.valueOf(f));
}

@Override
public void writeNumber(BigDecimal value)
   throws IOException, JsonGenerationException
{
   // Don't really know max length for big decimal, no point checking
   _verifyValueWrite("write number");
   if (value == null) {
      _writeNull();
   } else if (_cfgNumbersAsStrings) {
      _writeQuotedRaw(value);
   } else {
      writeRaw(value.toString());
   }
```

```
}

@Override
public void writeNumber(String encodedValue)
   throws IOException, JsonGenerationException
{
   _verifyValueWrite("write number");
   if (_cfgNumbersAsStrings) {
      _writeQuotedRaw(encodedValue);
   } else {
      writeRaw(encodedValue);
   }
}

private final void _writeQuotedRaw(Object value) throws IOException
{
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
   }
   _outputBuffer[_outputTail++] = '"';
   writeRaw(value.toString());
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
   }
   _outputBuffer[_outputTail++] = '"';
}

@Override
public void writeBoolean(boolean state)
   throws IOException, JsonGenerationException
{
   _verifyValueWrite("write boolean value");
   if ((_outputTail + 5) >= _outputEnd) {
      _flushBuffer();
   }
   int ptr = _outputTail;
   char[] buf = _outputBuffer;
   if (state) {
      buf[ptr] = 't';
      buf[++ptr] = 'r';
      buf[++ptr] = 'u';
      buf[++ptr] = 'e';
   } else {
      buf[ptr] = 'f';
      buf[++ptr] = 'a';
      buf[++ptr] = 'l';
      buf[++ptr] = 's';
      buf[++ptr] = 'e';
```

```
    }
    _outputTail = ptr+1;
}

@Override
public void writeNull()
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write null value");
    _writeNull();
}

/*
/**********************************************************
/* Implementations for other methods
/**********************************************************
 */

@Override
protected final void _verifyValueWrite(String typeMsg)
    throws IOException, JsonGenerationException
{
    int status = _writeContext.writeValue();
    if (status == JsonWriteContext.STATUS_EXPECT_NAME) {
        _reportError("Can not "+typeMsg+", expecting field name");
    }
    if (_cfgPrettyPrinter == null) {
        char c;
        switch (status) {
        case JsonWriteContext.STATUS_OK_AFTER_COMMA:
            c = ',';
            break;
        case JsonWriteContext.STATUS_OK_AFTER_COLON:
            c = ':';
            break;
        case JsonWriteContext.STATUS_OK_AFTER_SPACE:
            c = ' ';
            break;
        case JsonWriteContext.STATUS_OK_AS_IS:
        default:
            return;
        }
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail] = c;
        ++_outputTail;
        return;
```

```
    }
    // Otherwise, pretty printer knows what to do...
    _verifyPrettyValueWrite(typeMsg, status);
}

protected final void _verifyPrettyValueWrite(String typeMsg, int status)
    throws IOException, JsonGenerationException
{
    // If we have a pretty printer, it knows what to do:
    switch (status) {
    case JsonWriteContext.STATUS_OK_AFTER_COMMA: // array
        _cfgPrettyPrinter.writeArrayValueSeparator(this);
        break;
    case JsonWriteContext.STATUS_OK_AFTER_COLON:
        _cfgPrettyPrinter.writeObjectFieldValueSeparator(this);
        break;
    case JsonWriteContext.STATUS_OK_AFTER_SPACE:
        _cfgPrettyPrinter.writeRootValueSeparator(this);
        break;
    case JsonWriteContext.STATUS_OK_AS_IS:
        // First entry, but of which context?
        if (_writeContext.inArray()) {
            _cfgPrettyPrinter.beforeArrayValues(this);
        } else if (_writeContext.inObject()) {
            _cfgPrettyPrinter.beforeObjectEntries(this);
        }
        break;
    default:
        _cantHappen();
        break;
    }
}

/*
/**********************************************************
/* Low-level output handling
/**********************************************************
 */

@Override
public final void flush()
    throws IOException
{
    _flushBuffer();
    if (_writer != null) {
        if (isEnabled(Feature.FLUSH_PASSED_TO_STREAM)) {
            _writer.flush();
        }
```

```
    }
}

@Override
public void close()
    throws IOException
{
    super.close();

    /* 05-Dec-2008, tatu: To add [JACKSON-27], need to close open
     *   scopes.
     */
    // First: let's see that we still have buffers...
    if (_outputBuffer != null
        && isEnabled(Feature.AUTO_CLOSE_JSON_CONTENT)) {
        while (true) {
            JsonStreamContext ctxt = getOutputContext();
            if (ctxt.inArray()) {
                writeEndArray();
            } else if (ctxt.inObject()) {
                writeEndObject();
            } else {
                break;
            }
        }
    }
    _flushBuffer();

    /* 25-Nov-2008, tatus: As per [JACKSON-16] we are not to call close()
     *   on the underlying Reader, unless we "own" it, or auto-closing
     *   feature is enabled.
     *   One downside: when using UTF8Writer, underlying buffer(s)
     *   may not be properly recycled if we don't close the writer.
     */
    if (_ioContext.isResourceManaged() || isEnabled(Feature.AUTO_CLOSE_TARGET)) {
        _writer.close();
    } else {
        // If we can't close it, we should at least flush
        _writer.flush();
    }
    // Internal buffer(s) generator has can now be released as well
    _releaseBuffers();
}

@Override
protected void _releaseBuffers()
{
    char[] buf = _outputBuffer;
```

```
    if (buf != null) {
        _outputBuffer = null;
        _ioContext.releaseConcatBuffer(buf);
    }
}

/*
/**********************************************************
/* Internal methods, low-level writing
/**********************************************************
 */

private void _writeString(String text)
    throws IOException, JsonGenerationException
{
    /* One check first: if String won't fit in the buffer, let's
     * segment writes. No point in extending buffer to huge sizes
     * (like if someone wants to include multi-megabyte base64
     * encoded stuff or such)
     */
    int len = text.length();
    if (len > _outputEnd) { // Let's reserve space for entity at begin/end
        _writeLongString(text);
        return;
    }

    // Ok: we know String will fit in buffer ok
    // But do we need to flush first?
    if ((_outputTail + len) > _outputEnd) {
        _flushBuffer();
    }
    text.getChars(0, len, _outputBuffer, _outputTail);

    // And then we'll need to verify need for escaping etc:
    int end = _outputTail + len;
    final int[] escCodes = CharTypes.getOutputEscapes();
    final int escLen = escCodes.length;

    output_loop:
    while (_outputTail < end) {
        // Fast loop for chars not needing escaping
        escape_loop:
        while (true) {
            char c = _outputBuffer[_outputTail];
            if (c < escLen && escCodes[c] != 0) {
                break escape_loop;
            }
            if (++_outputTail >= end) {
```

```
                break output_loop;
            }
        }

        // Ok, bumped into something that needs escaping.
        /* First things first: need to flush the buffer.
         * Inlined, as we don't want to lose tail pointer
         */
        int flushLen = (_outputTail - _outputHead);
        if (flushLen > 0) {
            _writer.write(_outputBuffer, _outputHead, flushLen);
        }
        /* In any case, tail will be the new start, so hopefully
         * we have room now.
         */
        {
            int escCode = escCodes[_outputBuffer[_outputTail]];
            ++_outputTail;
            int needLen = (escCode < 0) ? 6 : 2;
            // If not, need to call separate method (note: buffer is empty now)
            if (needLen > _outputTail) {
                _outputHead = _outputTail;
                _writeSingleEscape(escCode);
            } else {
                // But if it fits, can just prepend to buffer
                int ptr = _outputTail - needLen;
                _outputHead = ptr;
                _appendSingleEscape(escCode, _outputBuffer, ptr);
            }
        }
    }
}

/**
 * Method called to write "long strings", strings whose length exceeds
 * output buffer length.
 */
private void _writeLongString(String text)
    throws IOException, JsonGenerationException
{
    // First things first: let's flush the buffer to get some more room
    _flushBuffer();

    // Then we can write
    final int textLen = text.length();
    int offset = 0;
    do {
        int max = _outputEnd;
```

```
            int segmentLen = ((offset + max) > textLen)
                ? (textLen - offset) : max;
            text.getChars(offset, offset+segmentLen, _outputBuffer, 0);
            _writeSegment(segmentLen);
            offset += segmentLen;
        } while (offset < textLen);
    }
    /**
     * Method called to output textual context which has been copied
     * to the output buffer prior to call. If any escaping is needed,
     * it will also be handled by the method.
     *<p>
     * Note: when called, textual content to write is within output
     * buffer, right after buffered content (if any). That's why only
     * length of that text is passed, as buffer and offset are implied.
     */
    private final void _writeSegment(int end)
        throws IOException, JsonGenerationException
    {
        final int[] escCodes = CharTypes.getOutputEscapes();
        final int escLen = escCodes.length;

        int ptr = 0;

        output_loop:
        while (ptr < end) {
            // Fast loop for chars not needing escaping
            int start = ptr;
            while (true) {
                char c = _outputBuffer[ptr];
                if (c < escLen && escCodes[c] != 0) {
                    break;
                }
                if (++ptr >= end) {
                    break;
                }
            }

            // Ok, bumped into something that needs escaping.
            /* First things first: need to flush the buffer.
             * Inlined, as we don't want to lose tail pointer
             */
            int flushLen = (ptr - start);
            if (flushLen > 0) {
                _writer.write(_outputBuffer, start, flushLen);
                if (ptr >= end) {
                    break output_loop;
                }
```

```
        }
        /* In any case, tail will be the new start, so hopefully
         * we have room now.
         */
        {
            int escCode = escCodes[_outputBuffer[ptr]];
            ++ptr;
            int needLen = (escCode < 0) ? 6 : 2;
            // If not, need to call separate method (note: buffer is empty now)
            if (needLen > _outputTail) {
                _writeSingleEscape(escCode);
            } else {
                // But if it fits, can just prepend to buffer
                ptr -= needLen;
                _appendSingleEscape(escCode, _outputBuffer, ptr);
            }
        }
    }
}

/**
 * This method called when the string content is already in
 * a char buffer, and need not be copied for processing.
 */
private void _writeString(char[] text, int offset, int len)
    throws IOException, JsonGenerationException
{
    /* Let's just find longest spans of non-escapable
     * content, and for each see if it makes sense
     * to copy them, or write through
     */
    len += offset; // -> len marks the end from now on
    final int[] escCodes = CharTypes.getOutputEscapes();
    final int escLen = escCodes.length;
    while (offset < len) {
        int start = offset;

        while (true) {
            char c = text[offset];
            if (c < escLen && escCodes[c] != 0) {
                break;
            }
            if (++offset >= len) {
                break;
            }
        }

        // Short span? Better just copy it to buffer first:
```

```
            int newAmount = offset - start;
            if (newAmount < SHORT_WRITE) {
                // Note: let's reserve room for escaped char (up to 6 chars)
                if ((_outputTail + newAmount) > _outputEnd) {
                    _flushBuffer();
                }
                if (newAmount > 0) {
                    System.arraycopy(text, start, _outputBuffer, _outputTail, newAmount);
                    _outputTail += newAmount;
                }
            } else { // Nope: better just write through
                _flushBuffer();
                _writer.write(text, start, newAmount);
            }
            // Was this the end?
            if (offset >= len) { // yup
                break;
            }
            // Nope, need to escape the char.
            int escCode = escCodes[text[offset]];
            ++offset;
            int needLen = (escCode < 0) ? 6 : 2;
            if ((_outputTail + needLen) > _outputEnd) {
                _flushBuffer();
            }
            _appendSingleEscape(escCode, _outputBuffer, _outputTail);
            _outputTail += needLen;
        }
    }

    protected void _writeBinary(Base64Variant b64variant, byte[] input, int inputPtr, final int inputEnd)
        throws IOException, JsonGenerationException
    {
        // Encoding is by chunks of 3 input, 4 output chars, so:
        int safeInputEnd = inputEnd - 3;
        // Let's also reserve room for possible (and quoted) lf char each round
        int safeOutputEnd = _outputEnd - 6;
        int chunksBeforeLF = b64variant.getMaxLineLength() >> 2;

        // Ok, first we loop through all full triplets of data:
        while (inputPtr <= safeInputEnd) {
            if (_outputTail > safeOutputEnd) { // need to flush
                _flushBuffer();
            }
            // First, mash 3 bytes into lsb of 32-bit int
            int b24 = ((int) input[inputPtr++]) << 8;
            b24 |= ((int) input[inputPtr++]) & 0xFF;
            b24 = (b24 << 8) | (((int) input[inputPtr++]) & 0xFF);
```

```
            _outputTail = b64variant.encodeBase64Chunk(b24, _outputBuffer, _outputTail);
            if (--chunksBeforeLF <= 0) {
                // note: must quote in JSON value
                _outputBuffer[_outputTail++] = '\\';
                _outputBuffer[_outputTail++] = 'n';
                chunksBeforeLF = b64variant.getMaxLineLength() >> 2;
            }
        }

        // And then we may have 1 or 2 leftover bytes to encode
        int inputLeft = inputEnd - inputPtr; // 0, 1 or 2
        if (inputLeft > 0) { // yes, but do we have room for output?
            if (_outputTail > safeOutputEnd) { // don't really need 6 bytes but...
                _flushBuffer();
            }
            int b24 = ((int) input[inputPtr++]) << 16;
            if (inputLeft == 2) {
                b24 |= (((int) input[inputPtr++]) & 0xFF) << 8;
            }
            _outputTail = b64variant.encodeBase64Partial(b24, inputLeft, _outputBuffer, _outputTail);
        }
    }

    private final void _writeNull() throws IOException
    {
        if ((_outputTail + 4) >= _outputEnd) {
            _flushBuffer();
        }
        int ptr = _outputTail;
        char[] buf = _outputBuffer;
        buf[ptr] = 'n';
        buf[++ptr] = 'u';
        buf[++ptr] = 'l';
        buf[++ptr] = 'l';
        _outputTail = ptr+1;
    }

    /**
     * @param escCode Character code for escape sequence (\C); or -1
     *   to indicate a generic (\\uXXXX) sequence.
     */
    private void _writeSingleEscape(int escCode)
        throws IOException
    {
        char[] buf = _entityBuffer;
        if (buf == null) {
            buf = new char[6];
            buf[0] = '\\';
```

```
            buf[2] = '0';
            buf[3] = '0';
        }

        if (escCode < 0) { // control char, value -(char + 1)
            int value = -(escCode + 1);
            buf[1] = 'u';
            // We know it's a control char, so only the last 2 chars are non-0
            buf[4] = HEX_CHARS[value >> 4];
            buf[5] = HEX_CHARS[value & 0xF];
            _writer.write(buf, 0, 6);
        } else {
            buf[1] = (char) escCode;
            _writer.write(buf, 0, 2);
        }
    }

    private void _appendSingleEscape(int escCode, char[] buf, int ptr)
    {
        if (escCode < 0) { // control char, value -(char + 1)
            int value = -(escCode + 1);
            buf[ptr] = '\\';
            buf[++ptr] = 'u';
            // We know it's a control char, so only the last 2 chars are non-0
            buf[++ptr] = '0';
            buf[++ptr] = '0';
            buf[++ptr] = HEX_CHARS[value >> 4];
            buf[++ptr] = HEX_CHARS[value & 0xF];
        } else {
            buf[ptr] = '\\';
            buf[ptr+1] = (char) escCode;
        }
    }

    protected final void _flushBuffer() throws IOException
    {
        int len = _outputTail - _outputHead;
        if (len > 0) {
            int offset = _outputHead;
            _outputTail = _outputHead = 0;
            _writer.write(_outputBuffer, offset, len);
        }
    }
}
package org.codehaus.jackson.impl;

import java.io.IOException;
```

```
import org.codehaus.jackson.*;
import org.codehaus.jackson.JsonParser.Feature;
import org.codehaus.jackson.io.NumberInput;

/**
* Intermediate base class used by all Jackson {@link JsonParser}
* implementations, but does not add any additional fields that depend
* on particular method of obtaining input.
*
* @since 1.6
*
* @author Tatu Saloranta
*/
public abstract class JsonParserMinimalBase
  extends JsonParser
{
  // Control chars:
  protected final static int INT_TAB = '\t';
  protected final static int INT_LF = '\n';
  protected final static int INT_CR = '\r';
  protected final static int INT_SPACE = 0x0020;

  // Markup
  protected final static int INT_LBRACKET = '[';
  protected final static int INT_RBRACKET = ']';
  protected final static int INT_LCURLY = '{';
  protected final static int INT_RCURLY = '}';
  protected final static int INT_QUOTE = '"';
  protected final static int INT_BACKSLASH = '\\';
  protected final static int INT_SLASH = '/';
  protected final static int INT_COLON = ':';
  protected final static int INT_COMMA = ',';
  protected final static int INT_ASTERISK = '*';
  protected final static int INT_APOSTROPHE = '\'';

  // Letters we need
  protected final static int INT_b = 'b';
  protected final static int INT_f = 'f';
  protected final static int INT_n = 'n';
  protected final static int INT_r = 'r';
  protected final static int INT_t = 't';
  protected final static int INT_u = 'u';

  /*
  /**********************************************************
  /* Life-cycle
  /**********************************************************
   */
```

```
protected JsonParserMinimalBase() { }
protected JsonParserMinimalBase(int features) {
   super(features);
}

/*
/**********************************************************
/* Configuration overrides if any
/**********************************************************
 */

// from base class:

//public void enableFeature(Feature f)
//public void disableFeature(Feature f)
//public void setFeature(Feature f, boolean state)
//public boolean isFeatureEnabled(Feature f)

/*
/**********************************************************
/* JsonParser impl
/**********************************************************
 */

@Override
public abstract JsonToken nextToken() throws IOException, JsonParseException;

//public final JsonToken nextValue()

@Override
public JsonParser skipChildren() throws IOException, JsonParseException
{
   if (_currToken != JsonToken.START_OBJECT
      && _currToken != JsonToken.START_ARRAY) {
      return this;
   }
   int open = 1;

   /* Since proper matching of start/end markers is handled
    * by nextToken(), we'll just count nesting levels here
    */
   while (true) {
      JsonToken t = nextToken();
      if (t == null) {
         _handleEOF();
         /* given constraints, above should never return;
          * however, FindBugs doesn't know about it and
```

```
             * complains... so let's add dummy break here
             */
            return this;
        }
        switch (t) {
        case START_OBJECT:
        case START_ARRAY:
            ++open;
            break;
        case END_OBJECT:
        case END_ARRAY:
            if (--open == 0) {
                return this;
            }
            break;
        }
    }
}

/**
 * Method sub-classes need to implement
 */
protected abstract void _handleEOF() throws JsonParseException;

//public JsonToken getCurrentToken()

//public boolean hasCurrentToken()

@Override
public abstract String getCurrentName() throws IOException, JsonParseException;

@Override
public abstract void close() throws IOException;

@Override
public abstract boolean isClosed();

@Override
public abstract JsonStreamContext getParsingContext();

//    public abstract JsonLocation getTokenLocation();

//   public abstract JsonLocation getCurrentLocation();

/*
/**********************************************************
/* Public API, access to token information, text
/**********************************************************
```

```
 */

@Override
public abstract String getText() throws IOException, JsonParseException;

@Override
public abstract char[] getTextCharacters() throws IOException, JsonParseException;

@Override
public abstract boolean hasTextCharacters();

@Override
public abstract int getTextLength() throws IOException, JsonParseException;

@Override
public abstract int getTextOffset() throws IOException, JsonParseException;

/*
/**********************************************************
/* Public API, access to token information, binary
/**********************************************************
 */

@Override
public abstract byte[] getBinaryValue(Base64Variant b64variant)
   throws IOException, JsonParseException;

/*
/**********************************************************
/* Public API, access with conversion/coercion
/**********************************************************
 */

@Override
public boolean getValueAsBoolean(boolean defaultValue) throws IOException, JsonParseException
{
   if (_currToken != null) {
      switch (_currToken) {
      case VALUE_NUMBER_INT:
         return getIntValue() != 0;
      case VALUE_TRUE:
         return true;
      case VALUE_FALSE:
      case VALUE_NULL:
         return false;
      case VALUE_EMBEDDED_OBJECT:
         {
            Object value = this.getEmbeddedObject();
```

```
            if (value instanceof Boolean) {
                return ((Boolean) value).booleanValue();
            }
        }
    case VALUE_STRING:
        String str = getText().trim();
        if ("true".equals(str)) {
            return true;
        }
        break;
    }
  }
  return defaultValue;
}

@Override
public int getValueAsInt(int defaultValue) throws IOException, JsonParseException
{
  if (_currToken != null) {
    switch (_currToken) {
    case VALUE_NUMBER_INT:
    case VALUE_NUMBER_FLOAT:
        return getIntValue();
    case VALUE_TRUE:
        return 1;
    case VALUE_FALSE:
    case VALUE_NULL:
        return 0;
    case VALUE_STRING:
        return NumberInput.parseAsInt(getText(), defaultValue);
    case VALUE_EMBEDDED_OBJECT:
        {
            Object value = this.getEmbeddedObject();
            if (value instanceof Number) {
                return ((Number) value).intValue();
            }
        }
    }
  }
  return defaultValue;
}

@Override
public long getValueAsLong(long defaultValue) throws IOException, JsonParseException
{
  if (_currToken != null) {
    switch (_currToken) {
    case VALUE_NUMBER_INT:
```

```
        case VALUE_NUMBER_FLOAT:
            return getLongValue();
        case VALUE_TRUE:
            return 1;
        case VALUE_FALSE:
        case VALUE_NULL:
            return 0;
        case VALUE_STRING:
            return NumberInput.parseAsLong(getText(), defaultValue);
        case VALUE_EMBEDDED_OBJECT:
            {
                Object value = this.getEmbeddedObject();
                if (value instanceof Number) {
                    return ((Number) value).longValue();
                }
            }
        }
    }
    return defaultValue;
}

@Override
public double getValueAsDouble(double defaultValue) throws IOException, JsonParseException
{
    if (_currToken != null) {
        switch (_currToken) {
        case VALUE_NUMBER_INT:
        case VALUE_NUMBER_FLOAT:
            return getDoubleValue();
        case VALUE_TRUE:
            return 1;
        case VALUE_FALSE:
        case VALUE_NULL:
            return 0;
        case VALUE_STRING:
            return NumberInput.parseAsDouble(getText(), defaultValue);
        case VALUE_EMBEDDED_OBJECT:
            {
                Object value = this.getEmbeddedObject();
                if (value instanceof Number) {
                    return ((Number) value).doubleValue();
                }
            }
        }
    }
    return defaultValue;
}
```

```
    /*
    /**********************************************************
    /* Error reporting
    /**********************************************************
     */

    protected void _reportUnexpectedChar(int ch, String comment)
        throws JsonParseException
    {
        String msg = "Unexpected character ("+_getCharDesc(ch)+")";
        if (comment != null) {
            msg += ": "+comment;
        }
        _reportError(msg);
    }

    protected void _reportInvalidEOF()
        throws JsonParseException
    {
        _reportInvalidEOF(" in "+_currToken);
    }

    protected void _reportInvalidEOF(String msg)
        throws JsonParseException
    {
        _reportError("Unexpected end-of-input"+msg);
    }

    protected void _throwInvalidSpace(int i)
        throws JsonParseException
    {
        char c = (char) i;
        String msg = "Illegal character ("+_getCharDesc(c)+"): only regular white space (\\r, \\n, \\t) is allowed between
tokens";
        _reportError(msg);
    }

    /**
     * Method called to report a problem with unquoted control character.
     * Note: starting with version 1.4, it is possible to suppress
     * exception by enabling {@link Feature#ALLOW_UNQUOTED_CONTROL_CHARS}.
     */
    protected void _throwUnquotedSpace(int i, String ctxtDesc)
        throws JsonParseException
    {
        // JACKSON-208; possible to allow unquoted control chars:
        if (!isEnabled(Feature.ALLOW_UNQUOTED_CONTROL_CHARS) || i >= INT_SPACE) {
            char c = (char) i;
```

```
            String msg = "Illegal unquoted character ("+_getCharDesc(c)+"): has to be escaped using backslash to be included in "+ctxtDesc;
            _reportError(msg);
        }
    }

    protected char _handleUnrecognizedCharacterEscape(char ch) throws JsonProcessingException
    {
        // as per [JACKSON-300]
        if (!isEnabled(Feature.ALLOW_BACKSLASH_ESCAPING_ANY_CHARACTER))  {
            _reportError("Unrecognized character escape "+_getCharDesc(ch));
        }
        return ch;
    }

    /*
    /**********************************************************
    /* Error reporting, generic
    /**********************************************************
     */

    protected final static String _getCharDesc(int ch)
    {
        char c = (char) ch;
        if (Character.isISOControl(c)) {
            return "(CTRL-CHAR, code "+ch+")";
        }
        if (ch > 255) {
            return "'"+c+"' (code "+ch+" / 0x"+Integer.toHexString(ch)+")";
        }
        return "'"+c+"' (code "+ch+")";
    }

    protected final void _reportError(String msg)
        throws JsonParseException
    {
        throw _constructError(msg);
    }

    protected final void _wrapError(String msg, Throwable t)
        throws JsonParseException
    {
        throw _constructError(msg, t);
    }

    protected final void _throwInternal()
    {
        throw new RuntimeException("Internal error: this code path should never get executed");
```

```
  }

  protected final JsonParseException _constructError(String msg, Throwable t)
  {
     return new JsonParseException(msg, getCurrentLocation(), t);
  }
}
package org.codehaus.jackson.impl;

import java.io.*;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.IOContext;
import org.codehaus.jackson.io.NumberOutput;
import org.codehaus.jackson.io.SerializedString;
import org.codehaus.jackson.util.CharTypes;

public class Utf8Generator
  extends JsonGeneratorBase
{
  private final static byte BYTE_u = (byte) 'u';

  private final static byte BYTE_0 = (byte) '0';

  private final static byte BYTE_LBRACKET = (byte) '[';
  private final static byte BYTE_RBRACKET = (byte) ']';
  private final static byte BYTE_LCURLY = (byte) '{';
  private final static byte BYTE_RCURLY = (byte) '}';

  private final static byte BYTE_BACKSLASH = (byte) '\\';
  private final static byte BYTE_SPACE = (byte) ' ';
  private final static byte BYTE_COMMA = (byte) ',';
  private final static byte BYTE_COLON = (byte) ':';
  private final static byte BYTE_QUOTE = (byte) '"';

  protected final static int SURR1_FIRST = 0xD800;
  protected final static int SURR1_LAST = 0xDBFF;
  protected final static int SURR2_FIRST = 0xDC00;
  protected final static int SURR2_LAST = 0xDFFF;

  // intermediate copies only made up to certain length...
  private final static int MAX_BYTES_TO_BUFFER = 512;

  final static byte[] HEX_CHARS = CharTypes.copyHexBytes();

  private final static byte[] NULL_BYTES = { 'n', 'u', 'l', 'l' };
```

```
private final static byte[] TRUE_BYTES = { 't', 'r', 'u', 'e' };
private final static byte[] FALSE_BYTES = { 'f', 'a', 'l', 's', 'e' };

private final static int[] sOutputEscapes = CharTypes.getOutputEscapes();

/*
/**********************************************************
/* Configuration
/**********************************************************
 */

final protected IOContext _ioContext;

final protected OutputStream _outputStream;

/*
/**********************************************************
/* Output buffering
/**********************************************************
 */

/**
 * Intermediate buffer in which contents are buffered before
 * being written using {@link #_outputStream}.
 */
protected byte[] _outputBuffer;

/**
 * Pointer to the position right beyond the last character to output
 * (end marker; may be past the buffer)
 */
protected int _outputTail = 0;

/**
 * End marker of the output buffer; one past the last valid position
 * within the buffer.
 */
protected final int _outputEnd;

/**
 * Maximum number of <code>char</code>s that we know will always fit
 * in the output buffer after escaping
 */
protected final int _outputMaxContiguous;

/**
 * Intermediate buffer in which characters of a String are copied
 * before being encoded.
```

```
 */
protected char[] _charBuffer;

/**
 * Length of <code>_charBuffer</code>
 */
protected final int _charBufferLength;

/**
 * 6 character temporary buffer allocated if needed, for constructing
 * escape sequences
 */
protected byte[] _entityBuffer;

/**
 * Flag that indicates whether the output buffer is recycable (and
 * needs to be returned to recycler once we are done) or not.
 */
protected boolean _bufferRecyclable;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public Utf8Generator(IOContext ctxt, int features, ObjectCodec codec,
        OutputStream out)
{

    super(features, codec);
    _ioContext = ctxt;
    _outputStream = out;
    _bufferRecyclable = true;
    _outputBuffer = ctxt.allocWriteEncodingBuffer();
    _outputEnd = _outputBuffer.length;
    /* To be exact, each char can take up to 6 bytes when escaped (Unicode
     * escape with backslash, 'u' and 4 hex digits); but to avoid fluctuation,
     * we will actually round down to only do up to 1/8 number of chars
     */
    _outputMaxContiguous = _outputEnd >> 3;
    _charBuffer = ctxt.allocConcatBuffer();
    _charBufferLength = _charBuffer.length;
}

public Utf8Generator(IOContext ctxt, int features, ObjectCodec codec,
        OutputStream out, byte[] outputBuffer, int outputOffset, boolean bufferRecyclable)
{
```

```
    super(features, codec);
    _ioContext = ctxt;
    _outputStream = out;
    _bufferRecyclable = bufferRecyclable;
    _outputTail = outputOffset;
    _outputBuffer = outputBuffer;
    _outputEnd = _outputBuffer.length;
    // up to 6 bytes per char (see above), rounded up to 1/8
    _outputMaxContiguous = _outputEnd >> 3;
    _charBuffer = ctxt.allocConcatBuffer();
    _charBufferLength = _charBuffer.length;
}

/*
/**********************************************************
/* Overridden methods
/**********************************************************
 */

/* Most overrides in this section are just to make methods final,
 * to allow better inlining...
 */
@Override
public final void writeStringField(String fieldName, String value)
    throws IOException, JsonGenerationException
{
    writeFieldName(fieldName);
    writeString(value);
}

@Override
public final void writeFieldName(String name)  throws IOException, JsonGenerationException
{
    int status = _writeContext.writeFieldName(name);
    if (status == JsonWriteContext.STATUS_EXPECT_VALUE) {
        _reportError("Can not write a field name, expecting a value");
    }
    if (_cfgPrettyPrinter != null) {
        _writePPFieldName(name, (status == JsonWriteContext.STATUS_OK_AFTER_COMMA));
        return;
    }
    if (status == JsonWriteContext.STATUS_OK_AFTER_COMMA) { // need comma
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = BYTE_COMMA;
    }
```

```
    _writeFieldName(name);
}

@Override
public final void writeFieldName(SerializedString name)
    throws IOException, JsonGenerationException
{
    // Object is a value, need to verify it's allowed
    int status = _writeContext.writeFieldName(name.getValue());
    if (status == JsonWriteContext.STATUS_EXPECT_VALUE) {
        _reportError("Can not write a field name, expecting a value");
    }
    if (_cfgPrettyPrinter != null) {
        _writePPFieldName(name, (status == JsonWriteContext.STATUS_OK_AFTER_COMMA));
        return;
    }
    if (status == JsonWriteContext.STATUS_OK_AFTER_COMMA) {
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = BYTE_COMMA;
    }
    _writeFieldName(name);
}

@Override
public final void writeFieldName(SerializableString name)
    throws IOException, JsonGenerationException
{
    // Object is a value, need to verify it's allowed
    int status = _writeContext.writeFieldName(name.getValue());
    if (status == JsonWriteContext.STATUS_EXPECT_VALUE) {
        _reportError("Can not write a field name, expecting a value");
    }
    if (_cfgPrettyPrinter != null) {
        _writePPFieldName(name, (status == JsonWriteContext.STATUS_OK_AFTER_COMMA));
        return;
    }
    if (status == JsonWriteContext.STATUS_OK_AFTER_COMMA) {
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = BYTE_COMMA;
    }
    _writeFieldName(name);
}

/*
```

```
/**********************************************************
/* Output method implementations, structural
/**********************************************************
 */

@Override
public final void writeStartArray() throws IOException, JsonGenerationException
{
   _verifyValueWrite("start an array");
   _writeContext = _writeContext.createChildArrayContext();
   if (_cfgPrettyPrinter != null) {
      _cfgPrettyPrinter.writeStartArray(this);
   } else {
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail++] = BYTE_LBRACKET;
   }
}

@Override
public final void writeEndArray() throws IOException, JsonGenerationException
{
   if (!_writeContext.inArray()) {
      _reportError("Current context not an ARRAY but "+_writeContext.getTypeDesc());
   }
   if (_cfgPrettyPrinter != null) {
      _cfgPrettyPrinter.writeEndArray(this, _writeContext.getEntryCount());
   } else {
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail++] = BYTE_RBRACKET;
   }
   _writeContext = _writeContext.getParent();
}

@Override
public final void writeStartObject() throws IOException, JsonGenerationException
{
   _verifyValueWrite("start an object");
   _writeContext = _writeContext.createChildObjectContext();
   if (_cfgPrettyPrinter != null) {
      _cfgPrettyPrinter.writeStartObject(this);
   } else {
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
```

```
        _outputBuffer[_outputTail++] = BYTE_LCURLY;
    }
}

@Override
public final void writeEndObject() throws IOException, JsonGenerationException
{
    if (!_writeContext.inObject()) {
        _reportError("Current context not an object but "+_writeContext.getTypeDesc());
    }
    _writeContext = _writeContext.getParent();
    if (_cfgPrettyPrinter != null) {
        _cfgPrettyPrinter.writeEndObject(this, _writeContext.getEntryCount());
    } else {
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = BYTE_RCURLY;
    }
}

protected final void _writeFieldName(String name)
    throws IOException, JsonGenerationException
{
    /* To support [JACKSON-46], we'll do this:
     * (Question: should quoting of spaces (etc) still be enabled?)
     */
    if (!isEnabled(Feature.QUOTE_FIELD_NAMES)) {
        _writeStringSegments(name);
        return;
    }
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    // The beef:
    final int len = name.length();
    if (len <= _charBufferLength) { // yes, fits right in
        name.getChars(0, len, _charBuffer, 0);
        // But as one segment, or multiple?
        if (len <= _outputMaxContiguous) {
            if ((_outputTail + len) > _outputEnd) { // caller must ensure enough space
                _flushBuffer();
            }
            _writeStringSegment(_charBuffer, 0, len);
        } else {
            _writeStringSegments(_charBuffer, 0, len);
        }
```

```
    } else {
        _writeStringSegments(name);
    }

    // and closing quotes; need room for one more char:
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

protected final void _writeFieldName(SerializableString name)
    throws IOException, JsonGenerationException
{
    byte[] raw = name.asQuotedUTF8();
    if (!isEnabled(Feature.QUOTE_FIELD_NAMES)) {
        _writeBytes(raw);
        return;
    }
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;

    // Can do it all in buffer?
    final int len = raw.length;
    if ((_outputTail + len + 1) < _outputEnd) { // yup
        System.arraycopy(raw, 0, _outputBuffer, _outputTail, len);
        _outputTail += len;
        _outputBuffer[_outputTail++] = BYTE_QUOTE;
    } else {
        _writeBytes(raw);
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = BYTE_QUOTE;
    }
}

/**
 * Specialized version of <code>_writeFieldName</code>, off-lined
 * to keep the "fast path" as simple (and hopefully fast) as possible.
 */
protected final void _writePPFieldName(String name, boolean commaBefore)
    throws IOException, JsonGenerationException
{
    if (commaBefore) {
        _cfgPrettyPrinter.writeObjectEntrySeparator(this);
```

```
    } else {
        _cfgPrettyPrinter.beforeObjectEntries(this);
    }

    if (isEnabled(Feature.QUOTE_FIELD_NAMES)) { // standard
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = BYTE_QUOTE;
        final int len = name.length();
        if (len <= _charBufferLength) { // yes, fits right in
            name.getChars(0, len, _charBuffer, 0);
            // But as one segment, or multiple?
            if (len <= _outputMaxContiguous) {
                if ((_outputTail + len) > _outputEnd) { // caller must ensure enough space
                    _flushBuffer();
                }
                _writeStringSegment(_charBuffer, 0, len);
            } else {
                _writeStringSegments(_charBuffer, 0, len);
            }
        } else {
            _writeStringSegments(name);
        }
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = BYTE_QUOTE;
    } else { // non-standard, omit quotes
        _writeStringSegments(name);
    }
}

protected final void _writePPFieldName(SerializableString name, boolean commaBefore)
    throws IOException, JsonGenerationException
{
    if (commaBefore) {
        _cfgPrettyPrinter.writeObjectEntrySeparator(this);
    } else {
        _cfgPrettyPrinter.beforeObjectEntries(this);
    }

    boolean addQuotes = isEnabled(Feature.QUOTE_FIELD_NAMES); // standard
    if (addQuotes) {
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = BYTE_QUOTE;
```

```
    }
    _writeBytes(name.asQuotedUTF8());
    if (addQuotes) {
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = BYTE_QUOTE;
    }
}

/*
/**********************************************************
/* Output method implementations, textual
/**********************************************************
 */

@Override
public void writeString(String text)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write text value");
    if (text == null) {
        _writeNull();
        return;
    }
    // First: can we make a local copy of chars that make up text?
    final int len = text.length();
    if (len > _charBufferLength) { // nope: off-line handling
        _writeLongString(text);
        return;
    }
    // yes: good.
    text.getChars(0, len, _charBuffer, 0);
    // Output: if we can't guarantee it fits in output buffer, off-line as well:
    if (len > _outputMaxContiguous) {
        _writeLongString(_charBuffer, 0, len);
        return;
    }
    if ((_outputTail + len + 2) > _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    _writeStringSegment(_charBuffer, 0, len); // we checked space already above
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

private final void _writeLongString(String text)
    throws IOException, JsonGenerationException
```

```
{
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    _writeStringSegments(text);
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

private final void _writeLongString(char[] text, int offset, int len)
    throws IOException, JsonGenerationException
{
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    _writeStringSegments(_charBuffer, 0, len);
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

@Override
public void writeString(char[] text, int offset, int len)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write text value");
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    // One or multiple segments?
    if (len <= _outputMaxContiguous) {
        if ((_outputTail + len) > _outputEnd) { // caller must ensure enough space
            _flushBuffer();
        }
        _writeStringSegment(text, offset, len);
    } else {
        _writeStringSegments(text, offset, len);
    }
    // And finally, closing quotes
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
```

```
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

@Override
public final void writeString(SerializableString text)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write text value");
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    _writeBytes(text.asQuotedUTF8());
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

@Override
public void writeRawUTF8String(byte[] text, int offset, int length)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write text value");
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    _writeBytes(text, offset, length);
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

@Override
public void writeUTF8String(byte[] text, int offset, int len)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write text value");
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    // One or multiple segments?
    if (len <= _outputMaxContiguous) {
        _writeUTF8Segment(text, offset, len);
    } else {
```

```
            _writeUTF8Segments(text, offset, len);
        }
        if (_outputTail >= _outputEnd) {
            _flushBuffer();
        }
        _outputBuffer[_outputTail++] = BYTE_QUOTE;
    }

    /*
    /**********************************************************
    /* Output method implementations, unprocessed ("raw")
    /**********************************************************
     */

    @Override
    public void writeRaw(String text)
        throws IOException, JsonGenerationException
    {
        int start = 0;
        int len = text.length();
        while (len > 0) {
            char[] buf = _charBuffer;
            final int blen = buf.length;
            final int len2 = (len < blen) ? len : blen;
            text.getChars(start, start+len2, buf, 0);
            writeRaw(buf, 0, len2);
            start += len2;
            len -= len2;
        }
    }

    @Override
    public void writeRaw(String text, int offset, int len)
        throws IOException, JsonGenerationException
    {
        while (len > 0) {
            char[] buf = _charBuffer;
            final int blen = buf.length;
            final int len2 = (len < blen) ? len : blen;
            text.getChars(offset, offset+len2, buf, 0);
            writeRaw(buf, 0, len2);
            offset += len2;
            len -= len2;
        }
    }

    // @TODO: rewrite for speed...
    @Override
```

```
public final void writeRaw(char[] cbuf, int offset, int len)
   throws IOException, JsonGenerationException
{
   // First: if we have 3 x charCount spaces, we know it'll fit just fine
   {
      int len3 = len+len+len;
      if ((_outputTail + len3) > _outputEnd) {
         // maybe we could flush?
         if (_outputEnd < len3) { // wouldn't be enough...
            _writeSegmentedRaw(cbuf, offset, len);
            return;
         }
         // yes, flushing brings enough space
         _flushBuffer();
      }
   }
   len += offset; // now marks the end

   // Note: here we know there is enough room, hence no output boundary checks
   main_loop:
   while (offset < len) {
      inner_loop:
      while (true) {
         int ch = (int) cbuf[offset];
         if (ch > 0x7F) {
            break inner_loop;
         }
         _outputBuffer[_outputTail++] = (byte) ch;
         if (++offset >= len) {
            break main_loop;
         }
      }
      char ch = cbuf[offset++];
      if (ch < 0x800) { // 2-byte?
         _outputBuffer[_outputTail++] = (byte) (0xc0 | (ch >> 6));
         _outputBuffer[_outputTail++] = (byte) (0x80 | (ch & 0x3f));
      } else {
         _outputRawMultiByteChar(ch, cbuf, offset, len);
      }
   }
}

@Override
public void writeRaw(char ch)
   throws IOException, JsonGenerationException
{
   if ((_outputTail + 3) >= _outputEnd) {
      _flushBuffer();
```

```
    }
    final byte[] bbuf = _outputBuffer;
    if (ch <= 0x7F) {
        bbuf[_outputTail++] = (byte) ch;
    } else  if (ch < 0x800) { // 2-byte?
        bbuf[_outputTail++] = (byte) (0xc0 | (ch >> 6));
        bbuf[_outputTail++] = (byte) (0x80 | (ch & 0x3f));
    } else {
        _outputRawMultiByteChar(ch, null, 0, 0);
    }
}

/**
 * Helper method called when it is possible that output of raw section
 * to output may cross buffer boundary
 */
private final void _writeSegmentedRaw(char[] cbuf, int offset, int len)
    throws IOException, JsonGenerationException
{
    final int end = _outputEnd;
    final byte[] bbuf = _outputBuffer;

    main_loop:
    while (offset < len) {
        inner_loop:
        while (true) {
            int ch = (int) cbuf[offset];
            if (ch >= 0x80) {
                break inner_loop;
            }
            // !!! TODO: fast(er) writes (roll input, output checks in one)
            if (_outputTail >= end) {
                _flushBuffer();
            }
            bbuf[_outputTail++] = (byte) ch;
            if (++offset >= len) {
                break main_loop;
            }
        }
        if ((_outputTail + 3) >= _outputEnd) {
            _flushBuffer();
        }
        char ch = cbuf[offset++];
        if (ch < 0x800) { // 2-byte?
            bbuf[_outputTail++] = (byte) (0xc0 | (ch >> 6));
            bbuf[_outputTail++] = (byte) (0x80 | (ch & 0x3f));
        } else {
            _outputRawMultiByteChar(ch, cbuf, offset, len);
```

```
        }
    }
}

/*
/**********************************************************
/* Output method implementations, base64-encoded binary
/**********************************************************
 */

@Override
public void writeBinary(Base64Variant b64variant, byte[] data, int offset, int len)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write binary value");
    // Starting quotes
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    _writeBinary(b64variant, data, offset, offset+len);
    // and closing quotes
    if (_outputTail >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

/*
/**********************************************************
/* Output method implementations, primitive
/**********************************************************
 */

@Override
public void writeNumber(int i)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write number");
    // up to 10 digits and possible minus sign
    if ((_outputTail + 11) >= _outputEnd) {
        _flushBuffer();
    }
    if (_cfgNumbersAsStrings) {
        _writeQuotedInt(i);
        return;
    }
    _outputTail = NumberOutput.outputInt(i, _outputBuffer, _outputTail);
```

```
}

private final void _writeQuotedInt(int i) throws IOException {
    if ((_outputTail + 13) >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    _outputTail = NumberOutput.outputInt(i, _outputBuffer, _outputTail);
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

@Override
public void writeNumber(long l)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write number");
    if (_cfgNumbersAsStrings) {
        _writeQuotedLong(l);
        return;
    }
    if ((_outputTail + 21) >= _outputEnd) {
        // up to 20 digits, minus sign
        _flushBuffer();
    }
    _outputTail = NumberOutput.outputLong(l, _outputBuffer, _outputTail);
}

private final void _writeQuotedLong(long l) throws IOException {
    if ((_outputTail + 23) >= _outputEnd) {
        _flushBuffer();
    }
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
    _outputTail = NumberOutput.outputLong(l, _outputBuffer, _outputTail);
    _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

@Override
public void writeNumber(BigInteger value)
    throws IOException, JsonGenerationException
{
    _verifyValueWrite("write number");
    if (value == null) {
        _writeNull();
    } else if (_cfgNumbersAsStrings) {
        _writeQuotedRaw(value);
    } else {
        writeRaw(value.toString());
    }
```

```
}


@Override
public void writeNumber(double d)
   throws IOException, JsonGenerationException
{
   if (_cfgNumbersAsStrings ||
      // [JACKSON-139]
      (((Double.isNaN(d) || Double.isInfinite(d))
         && isEnabled(Feature.QUOTE_NON_NUMERIC_NUMBERS)))) {
      writeString(String.valueOf(d));
      return;
   }
   // What is the max length for doubles? 40 chars?
   _verifyValueWrite("write number");
   writeRaw(String.valueOf(d));
}

@Override
public void writeNumber(float f)
   throws IOException, JsonGenerationException
{
   if (_cfgNumbersAsStrings ||
      // [JACKSON-139]
      (((Float.isNaN(f) || Float.isInfinite(f))
         && isEnabled(Feature.QUOTE_NON_NUMERIC_NUMBERS)))) {
      writeString(String.valueOf(f));
      return;
   }
   // What is the max length for floats?
   _verifyValueWrite("write number");
   writeRaw(String.valueOf(f));
}

@Override
public void writeNumber(BigDecimal value)
   throws IOException, JsonGenerationException
{
   // Don't really know max length for big decimal, no point checking
   _verifyValueWrite("write number");
   if (value == null) {
      _writeNull();
   } else if (_cfgNumbersAsStrings) {
      _writeQuotedRaw(value);
   } else {
      writeRaw(value.toString());
   }
```

```
}

@Override
public void writeNumber(String encodedValue)
   throws IOException, JsonGenerationException
{
   _verifyValueWrite("write number");
   if (_cfgNumbersAsStrings) {
      _writeQuotedRaw(encodedValue);
   } else {
      writeRaw(encodedValue);
   }
}

private final void _writeQuotedRaw(Object value) throws IOException
{
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
   }
   _outputBuffer[_outputTail++] = BYTE_QUOTE;
   writeRaw(value.toString());
   if (_outputTail >= _outputEnd) {
      _flushBuffer();
   }
   _outputBuffer[_outputTail++] = BYTE_QUOTE;
}

@Override
public void writeBoolean(boolean state)
   throws IOException, JsonGenerationException
{
   _verifyValueWrite("write boolean value");
   if ((_outputTail + 5) >= _outputEnd) {
      _flushBuffer();
   }
   byte[] keyword = state ? TRUE_BYTES : FALSE_BYTES;
   int len = keyword.length;
   System.arraycopy(keyword, 0, _outputBuffer, _outputTail, len);
   _outputTail += len;
}

@Override
public void writeNull()
   throws IOException, JsonGenerationException
{
   _verifyValueWrite("write null value");
   _writeNull();
}
```

```
/*
/**********************************************************
/* Implementations for other methods
/**********************************************************
 */

@Override
protected final void _verifyValueWrite(String typeMsg)
   throws IOException, JsonGenerationException
{
   int status = _writeContext.writeValue();
   if (status == JsonWriteContext.STATUS_EXPECT_NAME) {
      _reportError("Can not "+typeMsg+", expecting field name");
   }
   if (_cfgPrettyPrinter == null) {
      byte b;
      switch (status) {
      case JsonWriteContext.STATUS_OK_AFTER_COMMA:
         b = BYTE_COMMA;
         break;
      case JsonWriteContext.STATUS_OK_AFTER_COLON:
         b = BYTE_COLON;
         break;
      case JsonWriteContext.STATUS_OK_AFTER_SPACE:
         b = BYTE_SPACE;
         break;
      case JsonWriteContext.STATUS_OK_AS_IS:
      default:
         return;
      }
      if (_outputTail >= _outputEnd) {
         _flushBuffer();
      }
      _outputBuffer[_outputTail] = b;
      ++_outputTail;
      return;
   }
   // Otherwise, pretty printer knows what to do...
   _verifyPrettyValueWrite(typeMsg, status);
}

protected final void _verifyPrettyValueWrite(String typeMsg, int status)
   throws IOException, JsonGenerationException
{
   // If we have a pretty printer, it knows what to do:
   switch (status) {
   case JsonWriteContext.STATUS_OK_AFTER_COMMA: // array
```

```
            _cfgPrettyPrinter.writeArrayValueSeparator(this);
            break;
        case JsonWriteContext.STATUS_OK_AFTER_COLON:
            _cfgPrettyPrinter.writeObjectFieldValueSeparator(this);
            break;
        case JsonWriteContext.STATUS_OK_AFTER_SPACE:
            _cfgPrettyPrinter.writeRootValueSeparator(this);
            break;
        case JsonWriteContext.STATUS_OK_AS_IS:
            // First entry, but of which context?
            if (_writeContext.inArray()) {
                _cfgPrettyPrinter.beforeArrayValues(this);
            } else if (_writeContext.inObject()) {
                _cfgPrettyPrinter.beforeObjectEntries(this);
            }
            break;
        default:
            _cantHappen();
            break;
        }
    }

    /*
    /**********************************************************
    /* Low-level output handling
    /**********************************************************
     */

    @Override
    public final void flush()
        throws IOException
    {
        _flushBuffer();
        if (_outputStream != null) {
            if (isEnabled(Feature.FLUSH_PASSED_TO_STREAM)) {
                _outputStream.flush();
            }
        }
    }

    @Override
    public void close()
        throws IOException
    {
        super.close();

        /* 05-Dec-2008, tatu: To add [JACKSON-27], need to close open
         *   scopes.
```

```
     */
    // First: let's see that we still have buffers...
    if (_outputBuffer != null
        && isEnabled(Feature.AUTO_CLOSE_JSON_CONTENT)) {
        while (true) {
            JsonStreamContext ctxt = getOutputContext();
            if (ctxt.inArray()) {
                writeEndArray();
            } else if (ctxt.inObject()) {
                writeEndObject();
            } else {
                break;
            }
        }
    }
    _flushBuffer();

    /* 25-Nov-2008, tatus: As per [JACKSON-16] we are not to call close()
     *   on the underlying Reader, unless we "own" it, or auto-closing
     *   feature is enabled.
     *   One downside: when using UTF8Writer, underlying buffer(s)
     *   may not be properly recycled if we don't close the writer.
     */
    if (_ioContext.isResourceManaged() || isEnabled(Feature.AUTO_CLOSE_TARGET)) {
        _outputStream.close();
    } else {
        // If we can't close it, we should at least flush
        _outputStream.flush();
    }
    // Internal buffer(s) generator has can now be released as well
    _releaseBuffers();
}

@Override
protected void _releaseBuffers()
{
    byte[] buf = _outputBuffer;
    if (buf != null && _bufferRecyclable) {
        _outputBuffer = null;
        _ioContext.releaseWriteEncodingBuffer(buf);
    }
    char[] cbuf = _charBuffer;
    if (cbuf != null) {
        _charBuffer = null;
        _ioContext.releaseConcatBuffer(cbuf);
    }
}
```

```
/*
/**********************************************************
/* Internal methods, low-level writing
/**********************************************************
 */

private final void _writeBytes(byte[] bytes) throws IOException
{
    final int len = bytes.length;
    if ((_outputTail + len) > _outputEnd) {
        _flushBuffer();
        // still not enough?
        if (len > MAX_BYTES_TO_BUFFER) {
            _outputStream.write(bytes, 0, len);
            return;
        }
    }
    System.arraycopy(bytes, 0, _outputBuffer, _outputTail, len);
    _outputTail += len;
}

private final void _writeBytes(byte[] bytes, int offset, int len) throws IOException
{
    if ((_outputTail + len) > _outputEnd) {
        _flushBuffer();
        // still not enough?
        if (len > MAX_BYTES_TO_BUFFER) {
            _outputStream.write(bytes, offset, len);
            return;
        }
    }
    System.arraycopy(bytes, offset, _outputBuffer, _outputTail, len);
    _outputTail += len;
}

/**
 * Method called when String to write is long enough not to fit
 * completely in temporary copy buffer. If so, we will actually
 * copy it in small enough chunks so it can be directly fed
 * to single-segment writes (instead of maximum slices that
 * would fit in copy buffer)
 */
private final void _writeStringSegments(String text)
    throws IOException, JsonGenerationException
{
    int left = text.length();
    int offset = 0;
    final char[] cbuf = _charBuffer;
```

```
        while (left > 0) {
            int len = Math.min(_outputMaxContiguous, left);
            text.getChars(offset, offset+len, cbuf, 0);
            if ((_outputTail + len) > _outputEnd) { // caller must ensure enough space
                _flushBuffer();
            }
            _writeStringSegment(cbuf, 0, len);
            offset += len;
            left -= len;
        }
    }

    /**
     * Method called when character sequence to write is long enough that
     * its maximum encoded and escaped form is not guaranteed to fit in
     * the output buffer. If so, we will need to choose smaller output
     * chunks to write at a time.
     */
    private final void _writeStringSegments(char[] cbuf, int offset, int totalLen)
        throws IOException, JsonGenerationException
    {
        do {
            int len = Math.min(_outputMaxContiguous, totalLen);
            if ((_outputTail + len) > _outputEnd) { // caller must ensure enough space
                _flushBuffer();
            }
            _writeStringSegment(cbuf, offset, len);
            offset += len;
            totalLen -= len;
        } while (totalLen > 0);
    }

    /**
     * This method called when the string content is already in
     * a char buffer, and its maximum total encoded and escaped length
     * can not exceed size of the output buffer.
     * Caller must ensure that there is enough space in output buffer,
     * assuming case of all non-escaped ASCII characters, as well as
     * potentially enough space for other cases (but not necessarily flushed)
     */
    private final void _writeStringSegment(char[] cbuf, int offset, int len)
        throws IOException, JsonGenerationException
    {
        // note: caller MUST ensure (via flushing) there's room for ASCII only

        // Fast+tight loop for ASCII-only, no-escaping-needed output
        len += offset; // becomes end marker, then
```

```
    int outputPtr = _outputTail;
    final byte[] outputBuffer = _outputBuffer;
    final int[] escCodes = sOutputEscapes;

    while (offset < len) {
        int ch = cbuf[offset];
        if (ch > 0x7F || escCodes[ch] != 0) {
            break;
        }
        outputBuffer[outputPtr++] = (byte) ch;
        ++offset;
    }
    _outputTail = outputPtr;
    if (offset < len) {
        _writeStringSegment2(cbuf, offset, len);
    }
}

/**
 * Secondary method called when content contains characters to escape,
 * and/or multi-byte UTF-8 characters.
 */
private final void _writeStringSegment2(final char[] cbuf, int offset, final int end)
    throws IOException, JsonGenerationException
{
    // Ok: caller guarantees buffer can have room; but that may require flushing:
    if ((_outputTail +  6 * (end - offset)) > _outputEnd) {
        _flushBuffer();
    }

    int outputPtr = _outputTail;

    final byte[] outputBuffer = _outputBuffer;
    final int[] escCodes = sOutputEscapes;

    while (offset < end) {
        int ch = cbuf[offset++];
        if (ch <= 0x7F) {
            if (escCodes[ch] == 0) {
                outputBuffer[outputPtr++] = (byte) ch;
                continue;
            }
            int escape = escCodes[ch];
            if (escape > 0) { // 2-char escape, fine
                outputBuffer[outputPtr++] = BYTE_BACKSLASH;
                outputBuffer[outputPtr++] = (byte) escape;
            } else {
```

```
                // ctrl-char, 6-byte escape...
                outputPtr = _writeEscapedControlChar(escape, outputPtr);
            }
            continue;
        }
        if (ch <= 0x7FF) { // fine, just needs 2 byte output
            outputBuffer[outputPtr++] = (byte) (0xc0 | (ch >> 6));
            outputBuffer[outputPtr++] = (byte) (0x80 | (ch & 0x3f));
        } else {
            outputPtr = _outputMultiByteChar(ch, outputPtr);
        }
    }
    _outputTail = outputPtr;
}

/**
 * Method called when UTF-8 encoded (but NOT yet escaped!) content is not guaranteed
 * to fit in the output buffer after escaping; as such, we just need to
 * chunk writes.
 */
private final void _writeUTF8Segments(byte[] utf8, int offset, int totalLen)
    throws IOException, JsonGenerationException
{
    do {
        int len = Math.min(_outputMaxContiguous, totalLen);
        _writeUTF8Segment(utf8, offset, len);
        offset += len;
        totalLen -= len;
    } while (totalLen > 0);
}

private final void _writeUTF8Segment(byte[] utf8, final int offset, final int len)
    throws IOException, JsonGenerationException
{
    // fast loop to see if escaping is needed; don't copy, just look
    final int[] escCodes = sOutputEscapes;

    for (int ptr = offset, end = offset + len; ptr < end; ) {
        int ch = utf8[ptr++] & 0xFF;
        if (escCodes[ch] != 0) {
            _writeUTFSegment2(utf8, offset, len);
            return;
        }
    }

    // yes, fine, just copy the sucker
    if ((_outputTail + len) > _outputEnd) { // enough room or need to flush?
        _flushBuffer(); // but yes once we flush (caller guarantees length restriction)
```

```
    }
    System.arraycopy(utf8, offset, _outputBuffer, _outputTail, len);
    _outputTail += len;
}

private final void _writeUTFSegment2(final byte[] utf8, int offset, int len)
    throws IOException, JsonGenerationException
{
    int outputPtr = _outputTail;

    // Ok: caller guarantees buffer can have room; but that may require flushing:
    if ((outputPtr + (len * 6)) > _outputEnd) {
        _flushBuffer();
        outputPtr = _outputTail;
    }

    final byte[] outputBuffer = _outputBuffer;
    final int[] escCodes = sOutputEscapes;
    len += offset; // so 'len' becomes 'end'

    while (offset < len) {
        byte b = utf8[offset++];
        int ch = b & 0xFF;
        if (escCodes[ch] == 0) {
            outputBuffer[outputPtr++] = b;
            continue;
        }
        int escape = sOutputEscapes[ch];
        if (escape > 0) { // 2-char escape, fine
            outputBuffer[outputPtr++] = BYTE_BACKSLASH;
            outputBuffer[outputPtr++] = (byte) escape;
        } else {
            // ctrl-char, 6-byte escape...
            outputPtr = _writeEscapedControlChar(escape, outputPtr);
        }
    }
    _outputTail = outputPtr;
}

protected void _writeBinary(Base64Variant b64variant, byte[] input, int inputPtr, final int inputEnd)
    throws IOException, JsonGenerationException
{
    // Encoding is by chunks of 3 input, 4 output chars, so:
    int safeInputEnd = inputEnd - 3;
    // Let's also reserve room for possible (and quoted) lf char each round
    int safeOutputEnd = _outputEnd - 6;
    int chunksBeforeLF = b64variant.getMaxLineLength() >> 2;
```

```
    // Ok, first we loop through all full triplets of data:
    while (inputPtr <= safeInputEnd) {
        if (_outputTail > safeOutputEnd) { // need to flush
            _flushBuffer();
        }
        // First, mash 3 bytes into lsb of 32-bit int
        int b24 = ((int) input[inputPtr++]) << 8;
        b24 |= ((int) input[inputPtr++]) & 0xFF;
        b24 = (b24 << 8) | (((int) input[inputPtr++]) & 0xFF);
        _outputTail = b64variant.encodeBase64Chunk(b24, _outputBuffer, _outputTail);
        if (--chunksBeforeLF <= 0) {
            // note: must quote in JSON value
            _outputBuffer[_outputTail++] = '\\';
            _outputBuffer[_outputTail++] = 'n';
            chunksBeforeLF = b64variant.getMaxLineLength() >> 2;
        }
    }

    // And then we may have 1 or 2 leftover bytes to encode
    int inputLeft = inputEnd - inputPtr; // 0, 1 or 2
    if (inputLeft > 0) { // yes, but do we have room for output?
        if (_outputTail > safeOutputEnd) { // don't really need 6 bytes but...
            _flushBuffer();
        }
        int b24 = ((int) input[inputPtr++]) << 16;
        if (inputLeft == 2) {
            b24 |= (((int) input[inputPtr++]) & 0xFF) << 8;
        }
        _outputTail = b64variant.encodeBase64Partial(b24, inputLeft, _outputBuffer, _outputTail);
    }
}

/**
 * Method called to output a character that is beyond range of
 * 1- and 2-byte UTF-8 encodings, when outputting "raw"
 * text (meaning it is not to be escaped or quoted)
 */
private final int _outputRawMultiByteChar(int ch, char[] cbuf, int inputOffset, int inputLen)
    throws IOException
{
    // Let's handle surrogates gracefully (as 4 byte output):
    if (ch >= SURR1_FIRST) {
        if (ch <= SURR2_LAST) { // yes, outside of BMP
            // Do we have second part?
            if (inputOffset >= inputLen) { // nope... have to note down
                _reportError("Split surrogate on writeRaw() input (last character)");
            }
            _outputSurrogates(ch, cbuf[inputOffset]);
```

```
            return (inputOffset+1);
        }
    }
    final byte[] bbuf = _outputBuffer;
    bbuf[_outputTail++] = (byte) (0xe0 | (ch >> 12));
    bbuf[_outputTail++] = (byte) (0x80 | ((ch >> 6) & 0x3f));
    bbuf[_outputTail++] = (byte) (0x80 | (ch & 0x3f));
    return inputOffset;
}

protected final void _outputSurrogates(int surr1, int surr2)
    throws IOException
{
    int c = _decodeSurrogate(surr1, surr2);
    if ((_outputTail + 4) > _outputEnd) {
        _flushBuffer();
    }
    final byte[] bbuf = _outputBuffer;
    bbuf[_outputTail++] = (byte) (0xf0 | (c >> 18));
    bbuf[_outputTail++] = (byte) (0x80 | ((c >> 12) & 0x3f));
    bbuf[_outputTail++] = (byte) (0x80 | ((c >> 6) & 0x3f));
    bbuf[_outputTail++] = (byte) (0x80 | (c & 0x3f));
}

/**
 *
 * @param ch
 * @param outputPtr Position within output buffer to append multi-byte in
 *
 * @return New output position after appending
 *
 * @throws IOException
 */
private final int _outputMultiByteChar(int ch, int outputPtr)
    throws IOException
{
    byte[] bbuf = _outputBuffer;
    if (ch >= SURR1_FIRST && ch <= SURR2_LAST) { // yes, outside of BMP; add an escape
        bbuf[outputPtr++] = BYTE_BACKSLASH;
        bbuf[outputPtr++] = BYTE_u;

        bbuf[outputPtr++] = HEX_CHARS[(ch >> 12) & 0xF];
        bbuf[outputPtr++] = HEX_CHARS[(ch >> 8) & 0xF];
        bbuf[outputPtr++] = HEX_CHARS[(ch >> 4) & 0xF];
        bbuf[outputPtr++] = HEX_CHARS[ch & 0xF];
    } else {
        bbuf[outputPtr++] = (byte) (0xe0 | (ch >> 12));
        bbuf[outputPtr++] = (byte) (0x80 | ((ch >> 6) & 0x3f));
```

```
            bbuf[outputPtr++] = (byte) (0x80 | (ch & 0x3f));
        }
        return outputPtr;
    }

    protected final int _decodeSurrogate(int surr1, int surr2) throws IOException
    {
        // First is known to be valid, but how about the other?
        if (surr2 < SURR2_FIRST || surr2 > SURR2_LAST) {
            String msg = "Incomplete surrogate pair: first char 0x"+Integer.toHexString(surr1)+", second
0x"+Integer.toHexString(surr2);
            _reportError(msg);
        }
        int c = 0x10000 + ((surr1 - SURR1_FIRST) << 10) + (surr2 - SURR2_FIRST);
        return c;
    }

    private final void _writeNull() throws IOException
    {
        if ((_outputTail + 4) >= _outputEnd) {
            _flushBuffer();
        }
        System.arraycopy(NULL_BYTES, 0, _outputBuffer, _outputTail, 4);
        _outputTail += 4;
    }

    /**
     * @param escCode Character code for escape sequence (\C); or -1
     *   to indicate a generic (\\uXXXX) sequence.
     */
    private int _writeEscapedControlChar(int escCode, int outputPtr)
        throws IOException
    {
        final byte[] bbuf = _outputBuffer;
        bbuf[outputPtr++] = BYTE_BACKSLASH;
        int value = -(escCode + 1);
        bbuf[outputPtr++] = BYTE_u;
        bbuf[outputPtr++] = BYTE_0;
        bbuf[outputPtr++] = BYTE_0;
        // We know it's a control char, so only the last 2 chars are non-0
        bbuf[outputPtr++] = HEX_CHARS[value >> 4];
        bbuf[outputPtr++] = HEX_CHARS[value & 0xF];
        return outputPtr;
    }

    protected final void _flushBuffer() throws IOException
    {
        int len = _outputTail;
```

```
      if (len > 0) {
        _outputTail = 0;
        _outputStream.write(_outputBuffer, 0, len);
      }
    }
}
package org.codehaus.jackson.impl;

import java.io.*;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicLong;

import org.codehaus.jackson.*;
import org.codehaus.jackson.util.DefaultPrettyPrinter;
import org.codehaus.jackson.util.VersionUtil;

/**
* This base class implements part of API that a JSON generator exposes
* to applications, adds shared internal methods that sub-classes
* can use and adds some abstract methods sub-classes must implement.
*/
public abstract class JsonGeneratorBase
  extends JsonGenerator
{
  /*
  /**********************************************************
  /* Configuration
  /**********************************************************
   */

  protected ObjectCodec _objectCodec;

  /**
   * Bit flag composed of bits that indicate which
   * {@link org.codehaus.jackson.JsonGenerator.Feature}s
   * are enabled.
   */
  protected int _features;

  /**
   * Flag set to indicate that implicit conversion from number
   * to JSON String is needed (as per
   * {@link org.codehaus.jackson.JsonGenerator.Feature#WRITE_NUMBERS_AS_STRINGS}).
   */
  protected boolean _cfgNumbersAsStrings;
```

```
/*
/**********************************************************
/* State
/**********************************************************
 */

/**
 * Object that keeps track of the current contextual state
 * of the generator.
 */
protected JsonWriteContext _writeContext;

/**
 * Flag that indicates whether generator is closed or not. Gets
 * set when it is closed by an explicit call
 * ({@link #close}).
 */
protected boolean _closed;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

protected JsonGeneratorBase(int features, ObjectCodec codec)
{
   super();
   _features = features;
   _writeContext = JsonWriteContext.createRootContext();
   _objectCodec = codec;
   _cfgNumbersAsStrings = isEnabled(Feature.WRITE_NUMBERS_AS_STRINGS);
}

@Override
public Version version() {
   return VersionUtil.versionFor(getClass());
}

/*
/**********************************************************
/* Configuration
/**********************************************************
 */

@Override
public JsonGenerator enable(Feature f) {
```

```
    _features |= f.getMask();
    if (f == Feature.WRITE_NUMBERS_AS_STRINGS) {
        _cfgNumbersAsStrings = true;
    }
    return this;
}

@Override
public JsonGenerator disable(Feature f) {
    _features &= ~f.getMask();
    if (f == Feature.WRITE_NUMBERS_AS_STRINGS) {
        _cfgNumbersAsStrings = false;
    }
    return this;
}

//public JsonGenerator configure(Feature f, boolean state) { }

@Override
public final boolean isEnabled(Feature f) {
    return (_features & f.getMask()) != 0;
}

@Override
public JsonGenerator useDefaultPrettyPrinter()
{
    return setPrettyPrinter(new DefaultPrettyPrinter());
}

@Override
public JsonGenerator setCodec(ObjectCodec oc) {
    _objectCodec = oc;
    return this;
}

@Override
public final ObjectCodec getCodec() { return _objectCodec; }

/*
/**********************************************************
/* Public API, accessors
/**********************************************************
 */

/**
 * Note: co-variant return type.
 */
@Override
```

```
public final JsonWriteContext getOutputContext() { return _writeContext; }

/*
/**********************************************************
/* Public API, write methods, structural
/**********************************************************
 */

@Override
public void writeStartArray() throws IOException, JsonGenerationException
{
   // Array is a value, need to verify it's allowed
   _verifyValueWrite("start an array");
   _writeContext = _writeContext.createChildArrayContext();
   if (_cfgPrettyPrinter != null) {
      _cfgPrettyPrinter.writeStartArray(this);
   } else {
      _writeStartArray();
   }
}

/**
 * @deprecated since 1.7, should just override {@link #writeStartArray} instead
 *    of defining this method
 */
@Deprecated
protected void _writeStartArray() throws IOException, JsonGenerationException {
   // no-op, to be overridden
}

@Override
public void writeEndArray() throws IOException, JsonGenerationException
{
   if (!_writeContext.inArray()) {
      _reportError("Current context not an ARRAY but "+_writeContext.getTypeDesc());
   }
   if (_cfgPrettyPrinter != null) {
      _cfgPrettyPrinter.writeEndArray(this, _writeContext.getEntryCount());
   } else {
      _writeEndArray();
   }
   _writeContext = _writeContext.getParent();
}

/**
 * @deprecated since 1.7, should just override {@link #writeEndArray} instead
 *    of defining this method
 */
```

```
@Deprecated
protected void _writeEndArray() throws IOException, JsonGenerationException {
    // no-op, to be overridden
}

@Override
public void writeStartObject() throws IOException, JsonGenerationException
{
    _verifyValueWrite("start an object");
    _writeContext = _writeContext.createChildObjectContext();
    if (_cfgPrettyPrinter != null) {
        _cfgPrettyPrinter.writeStartObject(this);
    } else {
        _writeStartObject();
    }
}

/**
 * @deprecated since 1.7, should just override {@link #writeStartObject} instead
 *    of defining this method
 */
@Deprecated
protected void _writeStartObject() throws IOException, JsonGenerationException {
    // no-op, to be overridden
}

@Override
public void writeEndObject() throws IOException, JsonGenerationException
{
    if (!_writeContext.inObject()) {
        _reportError("Current context not an object but "+_writeContext.getTypeDesc());
    }
    _writeContext = _writeContext.getParent();
    if (_cfgPrettyPrinter != null) {
        _cfgPrettyPrinter.writeEndObject(this, _writeContext.getEntryCount());
    } else {
        _writeEndObject();
    }
}

/**
 * @deprecated since 1.7, should just override {@link #writeEndObject} instead
 *    of defining this method
 */
@Deprecated
protected void _writeEndObject() throws IOException, JsonGenerationException {
    // no-op, to be overridden
}
```

```
    /*
    /**********************************************************
    /* Public API, write methods, textual
    /**********************************************************
     */

    //public abstract void writeString(String text) throws IOException, JsonGenerationException;

    //public abstract void writeString(char[] text, int offset, int len) throws IOException, JsonGenerationException;

    //public abstract void writeRaw(String text) throws IOException, JsonGenerationException;

    //public abstract void writeRaw(char[] text, int offset, int len) throws IOException, JsonGenerationException;

    @Override
    public void writeRawValue(String text)
       throws IOException, JsonGenerationException
    {
       _verifyValueWrite("write raw value");
       writeRaw(text);
    }

    @Override
    public void writeRawValue(String text, int offset, int len)
       throws IOException, JsonGenerationException
    {
       _verifyValueWrite("write raw value");
       writeRaw(text, offset, len);
    }

    @Override
    public void writeRawValue(char[] text, int offset, int len)
       throws IOException, JsonGenerationException
    {
       _verifyValueWrite("write raw value");
       writeRaw(text, offset, len);
    }

    //public abstract void writeBinary(byte[] data, int offset, int len, boolean includeLFs) throws IOException,
JsonGenerationException;


    /*
    /**********************************************************
    /* Public API, write methods, primitive
    /**********************************************************
     */
```

```
// Not implemented at this level, added as placeholders

 /*
public abstract void writeNumber(int i)
public abstract void writeNumber(long l)
public abstract void writeNumber(double d)
public abstract void writeNumber(float f)
public abstract void writeNumber(BigDecimal dec)
public abstract void writeBoolean(boolean state)
public abstract void writeNull()
*/

/*
/**********************************************************
/* Public API, write methods, POJOs, trees
/**********************************************************
 */

@Override
public void writeObject(Object value)
   throws IOException, JsonProcessingException
{
   if (value == null) {
      // important: call method that does check value write:
      writeNull();
   } else {
      /* 02-Mar-2009, tatu: we are NOT to call _verifyValueWrite here,
       *   because that will be done when codec actually serializes
       *   contained POJO. If we did call it it would advance state
       *   causing exception later on
       */
      if (_objectCodec != null) {
         _objectCodec.writeValue(this, value);
         return;
      }
      _writeSimpleObject(value);
   }
}

@Override
public void writeTree(JsonNode rootNode)
   throws IOException, JsonProcessingException
{
   // As with 'writeObject()', we are not check if write would work
   if (rootNode == null) {
      writeNull();
   } else {
```

```
        if (_objectCodec == null) {
            throw new IllegalStateException("No ObjectCodec defined for the generator, can not serialize JsonNode-based trees");
        }
        _objectCodec.writeTree(this, rootNode);
    }
}

/*
/**********************************************************
/* Public API, low-level output handling
/**********************************************************
 */

@Override
public abstract void flush() throws IOException;

@Override
public void close() throws IOException
{
    _closed = true;
}

@Override
public boolean isClosed() { return _closed; }

/*
/**********************************************************
/* Public API, copy-through methods
/**********************************************************
 */

@Override
public final void copyCurrentEvent(JsonParser jp)
    throws IOException, JsonProcessingException
{
    JsonToken t = jp.getCurrentToken();
    // sanity check; what to do?
    if (t == null) {
        _reportError("No current event to copy");
    }
    switch(t) {
    case START_OBJECT:
        writeStartObject();
        break;
    case END_OBJECT:
        writeEndObject();
        break;
```

```
case START_ARRAY:
   writeStartArray();
   break;
case END_ARRAY:
   writeEndArray();
   break;
case FIELD_NAME:
   writeFieldName(jp.getCurrentName());
   break;
case VALUE_STRING:
   if (jp.hasTextCharacters()) {
      writeString(jp.getTextCharacters(), jp.getTextOffset(), jp.getTextLength());
   } else {
      writeString(jp.getText());
   }
   break;
case VALUE_NUMBER_INT:
   switch (jp.getNumberType()) {
   case INT:
      writeNumber(jp.getIntValue());
      break;
   case BIG_INTEGER:
      writeNumber(jp.getBigIntegerValue());
      break;
   default:
      writeNumber(jp.getLongValue());
   }
   break;
case VALUE_NUMBER_FLOAT:
   switch (jp.getNumberType()) {
   case BIG_DECIMAL:
      writeNumber(jp.getDecimalValue());
      break;
   case FLOAT:
      writeNumber(jp.getFloatValue());
      break;
   default:
      writeNumber(jp.getDoubleValue());
   }
   break;
case VALUE_TRUE:
   writeBoolean(true);
   break;
case VALUE_FALSE:
   writeBoolean(false);
   break;
case VALUE_NULL:
   writeNull();
```

```
            break;
        case VALUE_EMBEDDED_OBJECT:
            writeObject(jp.getEmbeddedObject());
            break;
        default:
            _cantHappen();
        }
    }

    @Override
    public final void copyCurrentStructure(JsonParser jp)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();

        // Let's handle field-name separately first
        if (t == JsonToken.FIELD_NAME) {
            writeFieldName(jp.getCurrentName());
            t = jp.nextToken();
            // fall-through to copy the associated value
        }

        switch (t) {
        case START_ARRAY:
            writeStartArray();
            while (jp.nextToken() != JsonToken.END_ARRAY) {
                copyCurrentStructure(jp);
            }
            writeEndArray();
            break;
        case START_OBJECT:
            writeStartObject();
            while (jp.nextToken() != JsonToken.END_OBJECT) {
                copyCurrentStructure(jp);
            }
            writeEndObject();
            break;
        default: // others are simple:
            copyCurrentEvent(jp);
        }
    }

    /*
    /**********************************************************
    /* Package methods for this, sub-classes
    /**********************************************************
     */
```

```
protected abstract void _releaseBuffers();

protected abstract void _verifyValueWrite(String typeMsg)
   throws IOException, JsonGenerationException;

protected void _reportError(String msg)
   throws JsonGenerationException
{
   throw new JsonGenerationException(msg);
}

protected void _cantHappen()
{
   throw new RuntimeException("Internal error: should never end up through this code path");
}

/**
 * Helper method to try to call appropriate write method for given
 * untyped Object. At this point, no structural conversions should be done,
 * only simple basic types are to be coerced as necessary.
 *
 * @param value Non-null value to write
 */
protected void _writeSimpleObject(Object value)
   throws IOException, JsonGenerationException
{
   /* 31-Dec-2009, tatu: Actually, we could just handle some basic
    *    types even without codec. This can improve interoperability,
    *    and specifically help with TokenBuffer.
    */
   if (value == null) {
      writeNull();
      return;
   }
   if (value instanceof String) {
      writeString((String) value);
      return;
   }
   if (value instanceof Number) {
      Number n = (Number) value;
      if (n instanceof Integer) {
         writeNumber(n.intValue());
         return;
      } else if (n instanceof Long) {
         writeNumber(n.longValue());
         return;
      } else if (n instanceof Double) {
         writeNumber(n.doubleValue());
```

```
            return;
        } else if (n instanceof Float) {
            writeNumber(n.floatValue());
            return;
        } else if (n instanceof Short) {
            writeNumber(n.shortValue());
            return;
        } else if (n instanceof Byte) {
            writeNumber(n.byteValue());
            return;
        } else if (n instanceof BigInteger) {
            writeNumber((BigInteger) n);
            return;
        } else if (n instanceof BigDecimal) {
            writeNumber((BigDecimal) n);
            return;

        // then Atomic types

        } else if (n instanceof AtomicInteger) {
            writeNumber(((AtomicInteger) n).get());
            return;
        } else if (n instanceof AtomicLong) {
            writeNumber(((AtomicLong) n).get());
            return;
        }
    } else if (value instanceof byte[]) {
        writeBinary((byte[]) value);
        return;
    } else if (value instanceof Boolean) {
        writeBoolean(((Boolean) value).booleanValue());
        return;
    } else if (value instanceof AtomicBoolean) {
        writeBoolean(((AtomicBoolean) value).get());
        return;
    }
    throw new IllegalStateException("No ObjectCodec defined for the generator, can only serialize simple wrapper
types (type passed "
            +value.getClass().getName()+")");
  }

  protected final void _throwInternal() {
    throw new RuntimeException("Internal error: this code path should never get executed");
  }

  /**
   * @since 1.7
   */
```

```
  protected void _reportUnsupportedOperation() {
     throw new UnsupportedOperationException("Operation not supported by generator of type
"+getClass().getName());
  }
}
package org.codehaus.jackson.impl;

/**
* Deprecated version of the default pretty printer.
*
* @deprecated Moved to {@link org.codehaus.jackson.util.DefaultPrettyPrinter}; will be removed in Jackson 2.0
*/
@Deprecated
public class DefaultPrettyPrinter
  extends org.codehaus.jackson.util.DefaultPrettyPrinter
{

}
package org.codehaus.jackson.impl;

import java.io.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.IOContext;
import org.codehaus.jackson.util.ByteArrayBuilder;
import org.codehaus.jackson.util.TextBuffer;
import org.codehaus.jackson.util.VersionUtil;

/**
* Intermediate base class used by all Jackson {@link JsonParser}
* implementations. Contains most common things that are independent
* of actual underlying input source
*
* @author Tatu Saloranta
*/
public abstract class JsonParserBase
  extends JsonParserMinimalBase
{
  /*
  /**********************************************************
  /* Generic I/O state
  /**********************************************************
   */

  /**
   * I/O context for this reader. It handles buffer allocation
   * for the reader.
   */
```

```
final protected IOContext _ioContext;

/**
 * Flag that indicates whether parser is closed or not. Gets
 * set when parser is either closed by explicit call
 * ({@link #close}) or when end-of-input is reached.
 */
protected boolean _closed;

/*
/**********************************************************
/* Current input data
/**********************************************************
 */

// Note: type of actual buffer depends on sub-class, can't include

/**
 * Pointer to next available character in buffer
 */
protected int _inputPtr = 0;

/**
 * Index of character after last available one in the buffer.
 */
protected int _inputEnd = 0;

/*
/**********************************************************
/* Current input location information
/**********************************************************
 */

/**
 * Number of characters that were contained in previous blocks
 * (blocks that were already processed prior to the current buffer).
 */
protected long _currInputProcessed = 0L;

/**
 * Current row location of current point in input buffer, starting
 * from 1
 */
protected int _currInputRow = 1;

/**
 * Current index of the first character of the current row in input
 * buffer. Needed to calculate column position, if necessary; benefit
```

```
     * of not having column itself is that this only has to be updated
     * once per line.
     */
    protected int _currInputRowStart = 0;

    /*
    /**********************************************************
    /* Information about starting location of event
    /* Reader is pointing to; updated on-demand
    /**********************************************************
     */

    // // // Location info at point when current token was started

    /**
     * Total number of characters read before start of current token.
     * For big (gigabyte-sized) sizes are possible, needs to be long,
     * unlike pointers and sizes related to in-memory buffers.
     */
    protected long _tokenInputTotal = 0;

    /**
     * Input row on which current token starts, 1-based
     */
    protected int _tokenInputRow = 1;

    /**
     * Column on input row that current token starts; 0-based (although
     * in the end it'll be converted to 1-based)
     */
    protected int _tokenInputCol = 0;

    /*
    /**********************************************************
    /* Parsing state
    /**********************************************************
     */

    /**
     * Information about parser context, context in which
     * the next token is to be parsed (root, array, object).
     */
    protected JsonReadContext _parsingContext;

    /**
     * Secondary token related to the next token after current one;
     * used if its type is known. This may be value token that
     * follows FIELD_NAME, for example.
```

```
 */
protected JsonToken _nextToken;

/*
/**********************************************************
/* Buffer(s) for local name(s) and text content
/**********************************************************
 */

/**
 * Buffer that contains contents of String values, including
 * field names if necessary (name split across boundary,
 * contains escape sequence, or access needed to char array)
 */
protected final TextBuffer _textBuffer;

/**
 * Temporary buffer that is needed if field name is accessed
 * using {@link #getTextCharacters} method (instead of String
 * returning alternatives)
 */
protected char[] _nameCopyBuffer = null;

/**
 * Flag set to indicate whether the field name is available
 * from the name copy buffer or not (in addition to its String
 * representation  being available via read context)
 */
protected boolean _nameCopied = false;

/**
 * ByteArrayBuilder is needed if 'getBinaryValue' is called. If so,
 * we better reuse it for remainder of content.
 */
protected ByteArrayBuilder _byteArrayBuilder = null;

/**
 * We will hold on to decoded binary data, for duration of
 * current event, so that multiple calls to
 * {@link #getBinaryValue} will not need to decode data more
 * than once.
 */
protected byte[] _binaryValue;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
```

```
 */

protected JsonParserBase(IOContext ctxt, int features)
{
   super();
   _features = features;
   _ioContext = ctxt;
   _textBuffer = ctxt.constructTextBuffer();
   _parsingContext = JsonReadContext.createRootContext(_tokenInputRow, _tokenInputCol);
}

@Override
public Version version() {
   return VersionUtil.versionFor(getClass());
}

/*
/**********************************************************
/* JsonParser impl
/**********************************************************
 */

/**
 * Method that can be called to get the name associated with
 * the current event.
 */
@Override
public String getCurrentName()
   throws IOException, JsonParseException
{
   // [JACKSON-395]: start markers require information from parent
   if (_currToken == JsonToken.START_OBJECT || _currToken == JsonToken.START_ARRAY) {
      JsonReadContext parent = _parsingContext.getParent();
      return parent.getCurrentName();
   }
   return _parsingContext.getCurrentName();
}

@Override
public void close() throws IOException
{
   if (!_closed) {
      _closed = true;
      try {
         _closeInput();
      } finally {
         // as per [JACKSON-324], do in finally block
         // Also, internal buffer(s) can now be released as well
```

```
            _releaseBuffers();
        }
    }
}

@Override
public boolean isClosed() { return _closed; }

@Override
public JsonReadContext getParsingContext()
{
    return _parsingContext;
}

/**
 * Method that return the <b>starting</b> location of the current
 * token; that is, position of the first character from input
 * that starts the current token.
 */
@Override
public JsonLocation getTokenLocation()
{
    return new JsonLocation(_ioContext.getSourceReference(),
                            getTokenCharacterOffset(),
                            getTokenLineNr(),
                            getTokenColumnNr());
}

/**
 * Method that returns location of the last processed character;
 * usually for error reporting purposes
 */
@Override
public JsonLocation getCurrentLocation()
{
    int col = _inputPtr - _currInputRowStart + 1; // 1-based
    return new JsonLocation(_ioContext.getSourceReference(),
                            _currInputProcessed + _inputPtr - 1,
                            _currInputRow, col);
}

/*
/**********************************************************
/* Public API, access to token information, text
/**********************************************************
 */
```

```
@Override
public boolean hasTextCharacters()
{
   if (_currToken != null) { // null only before/after document
      switch (_currToken) {
      case FIELD_NAME:
         return _nameCopied;
      case VALUE_STRING:
         return true; // usually true
      }
   }
   return false;
}

/*
/**********************************************************
/* Public low-level accessors
/**********************************************************
 */

public final long getTokenCharacterOffset() { return _tokenInputTotal; }
public final int getTokenLineNr() { return _tokenInputRow; }
public final int getTokenColumnNr() { return _tokenInputCol+1; }

/*
/**********************************************************
/* Low-level reading, other
/**********************************************************
 */

protected final void loadMoreGuaranteed()
   throws IOException
{
   if (!loadMore()) {
      _reportInvalidEOF();
   }
}

/*
/**********************************************************
/* Abstract methods needed from sub-classes
/**********************************************************
 */

protected abstract boolean loadMore() throws IOException;

protected abstract void _finishString() throws IOException, JsonParseException;
```

```
    protected abstract void _closeInput() throws IOException;

    protected abstract byte[] _decodeBase64(Base64Variant b64variant) throws IOException, JsonParseException;

    /*
    /**********************************************************
    /* Low-level reading, other
    /**********************************************************
     */

    /**
     * Method called to release internal buffers owned by the base
     * reader. This may be called along with {@link #_closeInput} (for
     * example, when explicitly closing this reader instance), or
     * separately (if need be).
     */
    protected void _releaseBuffers() throws IOException
    {
        _textBuffer.releaseBuffers();
        char[] buf = _nameCopyBuffer;
        if (buf != null) {
            _nameCopyBuffer = null;
            _ioContext.releaseNameCopyBuffer(buf);
        }
    }

    /**
     * Method called when an EOF is encountered between tokens.
     * If so, it may be a legitimate EOF, but only iff there
     * is no open non-root context.
     */
    @Override
    protected void _handleEOF() throws JsonParseException
    {
        if (!_parsingContext.inRoot()) {
            _reportInvalidEOF(": expected close marker for "+_parsingContext.getTypeDesc()+" (from
"+_parsingContext.getStartLocation(_ioContext.getSourceReference())+")");
        }
    }

    /*
    /**********************************************************
    /* Internal/package methods: Error reporting
    /**********************************************************
     */

    protected void _reportMismatchedEndMarker(int actCh, char expCh)
        throws JsonParseException
```

```
    {
        String startDesc = ""+_parsingContext.getStartLocation(_ioContext.getSourceReference());
        _reportError("Unexpected close marker '"+((char) actCh)+"': expected '"+expCh+"' (for
"+_parsingContext.getTypeDesc()+" starting at "+startDesc+")");
    }

    /*
    /**********************************************************
    /* Internal/package methods: shared/reusable builders
    /**********************************************************
     */

    public ByteArrayBuilder _getByteArrayBuilder()
    {
        if (_byteArrayBuilder == null) {
            _byteArrayBuilder = new ByteArrayBuilder();
        } else {
            _byteArrayBuilder.reset();
        }
        return _byteArrayBuilder;
    }

}
package org.codehaus.jackson.impl;

import java.io.IOException;

import org.codehaus.jackson.*;

/**
* Interface that defines objects that can produce indentation used
* to separate object entries and array values. Indentation in this
* context just means insertion of white space, independent of whether
* linefeeds are output.
*/
public interface Indenter
{
    public void writeIndentation(JsonGenerator jg, int level)
        throws IOException, JsonGenerationException;

    /**
     * @return True if indenter is considered inline (does not add linefeeds),
     *   false otherwise
     */
    public boolean isInline();
}
package org.codehaus.jackson.impl;
```

```
import org.codehaus.jackson.*;

/**
* Extension of {@link JsonStreamContext}, which implements
* core methods needed, and also exposes
* more complete API to generator implementation classes.
*/
public class JsonWriteContext
  extends JsonStreamContext
{
  // // // Return values for writeValue()

  public final static int STATUS_OK_AS_IS = 0;
  public final static int STATUS_OK_AFTER_COMMA = 1;
  public final static int STATUS_OK_AFTER_COLON = 2;
  public final static int STATUS_OK_AFTER_SPACE = 3; // in root context
  public final static int STATUS_EXPECT_VALUE = 4;
  public final static int STATUS_EXPECT_NAME = 5;

  protected final JsonWriteContext _parent;

  /**
   * Name of the field of which value is to be parsed; only
   * used for OBJECT contexts
   */
  protected String _currentName;

  /*
  /**********************************************************
  /* Simple instance reuse slots; speed up things
  /* a bit (10-15%) for docs with lots of small
  /* arrays/objects
  /**********************************************************
   */

  protected JsonWriteContext _child = null;

  /*
  /**********************************************************
  /* Life-cycle
  /**********************************************************
   */

  protected JsonWriteContext(int type, JsonWriteContext parent)
  {
     super();
     _type = type;
     _parent = parent;
```

```
    _index = -1;
}

// // // Factory methods

public static JsonWriteContext createRootContext()
{
    return new JsonWriteContext(TYPE_ROOT, null);
}

private final JsonWriteContext reset(int type) {
    _type = type;
    _index = -1;
    _currentName = null;
    return this;
}

public final JsonWriteContext createChildArrayContext()
{
    JsonWriteContext ctxt = _child;
    if (ctxt == null) {
        _child = ctxt = new JsonWriteContext(TYPE_ARRAY, this);
        return ctxt;
    }
    return ctxt.reset(TYPE_ARRAY);
}

public final JsonWriteContext createChildObjectContext()
{
    JsonWriteContext ctxt = _child;
    if (ctxt == null) {
        _child = ctxt = new JsonWriteContext(TYPE_OBJECT, this);
        return ctxt;
    }
    return ctxt.reset(TYPE_OBJECT);
}

// // // Shared API

@Override
public final JsonWriteContext getParent() { return _parent; }

@Override
public final String getCurrentName() { return _currentName; }

// // // API sub-classes are to implement

/**
```

```
     * Method that writer is to call before it writes a field name.
     *
     * @return Index of the field entry (0-based)
     */
    public final int writeFieldName(String name)
    {
        if (_type == TYPE_OBJECT) {
            if (_currentName != null) { // just wrote a name...
                return STATUS_EXPECT_VALUE;
            }
            _currentName = name;
            return (_index < 0) ? STATUS_OK_AS_IS : STATUS_OK_AFTER_COMMA;
        }
        return STATUS_EXPECT_VALUE;
    }

    public final int writeValue()
    {
        // Most likely, object:
        if (_type == TYPE_OBJECT) {
            if (_currentName == null) {
                return STATUS_EXPECT_NAME;
            }
            _currentName = null;
            ++_index;
            return STATUS_OK_AFTER_COLON;
        }

        // Ok, array?
        if (_type == TYPE_ARRAY) {
            int ix = _index;
            ++_index;
            return (ix < 0) ? STATUS_OK_AS_IS : STATUS_OK_AFTER_COMMA;
        }

        // Nope, root context
        // No commas within root context, but need space
        ++_index;
        return (_index == 0) ? STATUS_OK_AS_IS : STATUS_OK_AFTER_SPACE;
    }

    // // // Internally used abstract methods

    protected final void appendDesc(StringBuilder sb)
    {
        if (_type == TYPE_OBJECT) {
            sb.append('{');
            if (_currentName != null) {
```

```
                sb.append("");
                // !!! TODO: Name chars should be escaped?
                sb.append(_currentName);
                sb.append("");
            } else {
                sb.append('?');
            }
            sb.append(']');
        } else if (_type == TYPE_ARRAY) {
            sb.append('[');
            sb.append(getCurrentIndex());
            sb.append(']');
        } else {
            // nah, ROOT:
            sb.append("/");
        }
    }

    // // // Overridden standard methods

    /**
     * Overridden to provide developer writeable "JsonPath" representation
     * of the context.
     */
    @Override
    public final String toString()
    {
        StringBuilder sb = new StringBuilder(64);
        appendDesc(sb);
        return sb.toString();
    }
}
package org.codehaus.jackson.impl;

import java.io.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.*;
import org.codehaus.jackson.sym.BytesToNameCanonicalizer;
import org.codehaus.jackson.sym.CharsToNameCanonicalizer;

/**
* This class is used to determine the encoding of byte stream
* that is to contain JSON content. Rules are fairly simple, and
* defined in JSON specification (RFC-4627 or newer), except
* for BOM handling, which is a property of underlying
* streams.
*/
```

```
public final class ByteSourceBootstrapper
{
  /*
  /**********************************************************
  /* Configuration
  /**********************************************************
   */

  final IOContext _context;

  final InputStream _in;

  /*
  /**********************************************************
  /* Input buffering
  /**********************************************************
   */

  final byte[] _inputBuffer;

  private int _inputPtr;

  private int _inputEnd;

  /**
   * Flag that indicates whether buffer above is to be recycled
   * after being used or not.
   */
  private final boolean _bufferRecyclable;

  /*
  /**********************************************************
  /* Input location
  /**********************************************************
   */

  /**
   * Current number of input units (bytes or chars) that were processed in
   * previous blocks,
   * before contents of current input buffer.
   *<p>
   * Note: includes possible BOMs, if those were part of the input.
   */
  protected int _inputProcessed;

  /*
  /**********************************************************
  /* Data gathered
```

```
/**********************************************************
 */

boolean _bigEndian = true;
int _bytesPerChar = 0; // 0 means "dunno yet"

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public ByteSourceBootstrapper(IOContext ctxt, InputStream in)
{
    _context = ctxt;
    _in = in;
    _inputBuffer = ctxt.allocReadIOBuffer();
    _inputEnd = _inputPtr = 0;
    _inputProcessed = 0;
    _bufferRecyclable = true;
}

public ByteSourceBootstrapper(IOContext ctxt, byte[] inputBuffer, int inputStart, int inputLen)
{
    _context = ctxt;
    _in = null;
    _inputBuffer = inputBuffer;
    _inputPtr = inputStart;
    _inputEnd = (inputStart + inputLen);
    // Need to offset this for correct location info
    _inputProcessed = -inputStart;
    _bufferRecyclable = false;
}

/**
 * Method that should be called after constructing an instace.
 * It will figure out encoding that content uses, to allow
 * for instantiating a proper scanner object.
 */
public JsonEncoding detectEncoding()
    throws IOException, JsonParseException
{
    boolean foundEncoding = false;

    // First things first: BOM handling
    /* Note: we can require 4 bytes to be read, since no
     * combination of BOM + valid JSON content can have
     * shorter length (shortest valid JSON content is single
```

```
         * digit char, but BOMs are chosen such that combination
         * is always at least 4 chars long)
         */
        if (ensureLoaded(4)) {
            int quad =  (_inputBuffer[_inputPtr] << 24)
                | ((_inputBuffer[_inputPtr+1] & 0xFF) << 16)
                | ((_inputBuffer[_inputPtr+2] & 0xFF) << 8)
                | (_inputBuffer[_inputPtr+3] & 0xFF);

            if (handleBOM(quad)) {
                foundEncoding = true;
            } else {
                /* If no BOM, need to auto-detect based on first char;
                 * this works since it must be 7-bit ascii (wrt. unicode
                 * compatible encodings, only ones JSON can be transferred
                 * over)
                 */
                // UTF-32?
                if (checkUTF32(quad)) {
                    foundEncoding = true;
                } else if (checkUTF16(quad >>> 16)) {
                    foundEncoding = true;
                }
            }
        } else if (ensureLoaded(2)) {
            int i16 = ((_inputBuffer[_inputPtr] & 0xFF) << 8)
                | (_inputBuffer[_inputPtr+1] & 0xFF);
            if (checkUTF16(i16)) {
                foundEncoding = true;
            }
        }

        JsonEncoding enc;

        /* Not found yet? As per specs, this means it must be UTF-8. */
        if (!foundEncoding) {
            enc = JsonEncoding.UTF8;
        } else if (_bytesPerChar == 2) {
            enc = _bigEndian ? JsonEncoding.UTF16_BE : JsonEncoding.UTF16_LE;
        } else if (_bytesPerChar == 4) {
            enc = _bigEndian ? JsonEncoding.UTF32_BE : JsonEncoding.UTF32_LE;
        } else {
            throw new RuntimeException("Internal error"); // should never get here
        }
        _context.setEncoding(enc);
        return enc;
    }
```

```
  public Reader constructReader()
    throws IOException
  {
    JsonEncoding enc = _context.getEncoding();
    switch (enc) {
    case UTF32_BE:
    case UTF32_LE:
      return new UTF32Reader(_context, _in, _inputBuffer, _inputPtr, _inputEnd,
                             _context.getEncoding().isBigEndian());

    case UTF16_BE:
    case UTF16_LE:
    case UTF8: // only in non-common case where we don't want to do direct mapping
      {
        // First: do we have a Stream? If not, need to create one:
        InputStream in = _in;

        if (in == null) {
          in = new ByteArrayInputStream(_inputBuffer, _inputPtr, _inputEnd);
        } else {
          /* Also, if we have any read but unused input (usually true),
           * need to merge that input in:
           */
          if (_inputPtr < _inputEnd) {
            in = new MergedStream(_context, in, _inputBuffer, _inputPtr, _inputEnd);
          }
        }
        return new InputStreamReader(in, enc.getJavaName());
      }
    }
    throw new RuntimeException("Internal error"); // should never get here
  }

  public JsonParser constructParser(int features, ObjectCodec codec, BytesToNameCanonicalizer rootByteSymbols,
CharsToNameCanonicalizer rootCharSymbols)
    throws IOException, JsonParseException
  {
    JsonEncoding enc = detectEncoding();

    // As per [JACKSON-259], may want to fully disable canonicalization:
    boolean canonicalize = JsonParser.Feature.CANONICALIZE_FIELD_NAMES.enabledIn(features);
    boolean intern = JsonParser.Feature.INTERN_FIELD_NAMES.enabledIn(features);
    if (enc == JsonEncoding.UTF8) {
      /* and without canonicalization, byte-based approach is not performance; just use std UTF-8 reader
       * (which is ok for larger input; not so hot for smaller; but this is not a common case)
       */
      if (canonicalize) {
        BytesToNameCanonicalizer can = rootByteSymbols.makeChild(canonicalize, intern);
```

```
            return new Utf8StreamParser(_context, features, _in, codec, can, _inputBuffer, _inputPtr, _inputEnd,
_bufferRecyclable);
        }
    }
    return new ReaderBasedParser(_context, features, constructReader(), codec,
rootCharSymbols.makeChild(canonicalize, intern));
  }

  /*
  /**********************************************************
  /* Internal methods, parsing
  /**********************************************************
   */

  /**
   * @return True if a BOM was succesfully found, and encoding
   *   thereby recognized.
   */
  private boolean handleBOM(int quad)
    throws IOException
  {
    /* Handling of (usually) optional BOM (required for
     * multi-byte formats); first 32-bit charsets:
     */
    switch (quad) {
    case 0x0000FEFF:
      _bigEndian = true;
      _inputPtr += 4;
      _bytesPerChar = 4;
      return true;
    case 0xFFFE0000: // UCS-4, LE?
      _inputPtr += 4;
      _bytesPerChar = 4;
      _bigEndian = false;
      return true;
    case 0x0000FFFE: // UCS-4, in-order...
      reportWeirdUCS4("2143"); // throws exception
    case 0xFEFF0000: // UCS-4, in-order...
      reportWeirdUCS4("3412"); // throws exception
    }
    // Ok, if not, how about 16-bit encoding BOMs?
    int msw = quad >>> 16;
    if (msw == 0xFEFF) { // UTF-16, BE
      _inputPtr += 2;
      _bytesPerChar = 2;
      _bigEndian = true;
      return true;
    }
```

```
    if (msw == 0xFFFE) { // UTF-16, LE
        _inputPtr += 2;
        _bytesPerChar = 2;
        _bigEndian = false;
        return true;
    }
    // And if not, then UTF-8 BOM?
    if ((quad >>> 8) == 0xEFBBBF) { // UTF-8
        _inputPtr += 3;
        _bytesPerChar = 1;
        _bigEndian = true; // doesn't really matter
        return true;
    }
    return false;
}

private boolean checkUTF32(int quad)
    throws IOException
{
    /* Handling of (usually) optional BOM (required for
     * multi-byte formats); first 32-bit charsets:
     */
    if ((quad >> 8) == 0) { // 0x000000?? -> UTF32-BE
        _bigEndian = true;
    } else if ((quad & 0x00FFFFFF) == 0) { // 0x??000000 -> UTF32-LE
        _bigEndian = false;
    } else if ((quad & ~0x00FF0000) == 0) { // 0x00??0000 -> UTF32-in-order
        reportWeirdUCS4("3412");
    } else if ((quad & ~0x0000FF00) == 0) { // 0x0000??00 -> UTF32-in-order
        reportWeirdUCS4("2143");
    } else {
        // Can not be valid UTF-32 encoded JSON...
        return false;
    }
    // Not BOM (just regular content), nothing to skip past:
    //_inputPtr += 4;
    _bytesPerChar = 4;
    return true;
}

private boolean checkUTF16(int i16)
{
    if ((i16 & 0xFF00) == 0) { // UTF-16BE
        _bigEndian = true;
    } else if ((i16 & 0x00FF) == 0) { // UTF-16LE
        _bigEndian = false;
    } else { // nope, not  UTF-16
        return false;
```

```
    }
    // Not BOM (just regular content), nothing to skip past:
    //_inputPtr += 2;
    _bytesPerChar = 2;
    return true;
}

/*
/**********************************************************
/* Internal methods, problem reporting
/**********************************************************
 */

private void reportWeirdUCS4(String type)
    throws IOException
{
    throw new CharConversionException("Unsupported UCS-4 endianness ("+type+") detected");
}

/*
/**********************************************************
/* Internal methods, raw input access
/**********************************************************
 */

protected boolean ensureLoaded(int minimum)
    throws IOException
{
    /* Let's assume here buffer has enough room -- this will always
     * be true for the limited used this method gets
     */
    int gotten = (_inputEnd - _inputPtr);
    while (gotten < minimum) {
        int count;

        if (_in == null) { // block source
            count = -1;
        } else {
            count = _in.read(_inputBuffer, _inputEnd, _inputBuffer.length - _inputEnd);
        }
        if (count < 1) {
            return false;
        }
        _inputEnd += count;
        gotten += count;
    }
    return true;
}
```

```
}
/**
* Parser and generator implementation classes that Jackson
* defines and uses.
* Application code should not (need to) use contents of this package.
*/
package org.codehaus.jackson.impl;
package org.codehaus.jackson.impl;

import org.codehaus.jackson.*;
import org.codehaus.jackson.util.CharTypes;

/**
* Extension of {@link JsonStreamContext}, which implements
* core methods needed, and also exposes
* more complete API to parser implementation classes.
*/
public final class JsonReadContext
  extends JsonStreamContext
{
  // // // Configuration

  protected final JsonReadContext _parent;

  // // // Location information (minus source reference)

  protected int _lineNr;
  protected int _columnNr;

  protected String _currentName;

  /*
  /******************************************************
  /* Simple instance reuse slots; speeds up things
  /* a bit (10-15%) for docs with lots of small
  /* arrays/objects (for which allocation was
  /* visible in profile stack frames)
  /******************************************************
   */

  protected JsonReadContext _child = null;

  /*
  /******************************************************
  /* Instance construction, reuse
  /******************************************************
   */
```

```
public JsonReadContext(JsonReadContext parent, int type, int lineNr, int colNr)
{
   super();
   _type = type;
   _parent = parent;
   _lineNr = lineNr;
   _columnNr = colNr;
   _index = -1;
}

protected final void reset(int type, int lineNr, int colNr)
{
   _type = type;
   _index = -1;
   _lineNr = lineNr;
   _columnNr = colNr;
   _currentName = null;
}

// // // Factory methods

public static JsonReadContext createRootContext(int lineNr, int colNr)
{
   return new JsonReadContext(null, TYPE_ROOT, lineNr, colNr);
}

public final JsonReadContext createChildArrayContext(int lineNr, int colNr)
{
   JsonReadContext ctxt = _child;
   if (ctxt == null) {
      _child = ctxt = new JsonReadContext(this, TYPE_ARRAY, lineNr, colNr);
      return ctxt;
   }
   ctxt.reset(TYPE_ARRAY, lineNr, colNr);
   return ctxt;
}

public final JsonReadContext createChildObjectContext(int lineNr, int colNr)
{
   JsonReadContext ctxt = _child;
   if (ctxt == null) {
      _child = ctxt = new JsonReadContext(this, TYPE_OBJECT, lineNr, colNr);
      return ctxt;
   }
   ctxt.reset(TYPE_OBJECT, lineNr, colNr);
   return ctxt;
}
```

```
/*
/**********************************************************
/* Abstract method implementation
/**********************************************************
 */

@Override
public final String getCurrentName() { return _currentName; }

@Override
public final JsonReadContext getParent() { return _parent; }

/*
/**********************************************************
/* Extended API
/**********************************************************
 */

/**
 * @return Location pointing to the point where the context
 *   start marker was found
 */
public final JsonLocation getStartLocation(Object srcRef)
{
   /* We don't keep track of offsets at this level (only
    * reader does)
    */
   long totalChars = -1L;

   return new JsonLocation(srcRef, totalChars, _lineNr, _columnNr);
}

/*
/**********************************************************
/* State changes
/**********************************************************
 */

public final boolean expectComma()
{
   /* Assumption here is that we will be getting a value (at least
    * before calling this method again), and
    * so will auto-increment index to avoid having to do another call
    */
   int ix = ++_index; // starts from -1
   return (_type != TYPE_ROOT && ix > 0);
}
```

```
  public void setCurrentName(String name)
  {
     _currentName = name;
  }

  /*
  /**********************************************************
  /* Overridden standard methods
  /**********************************************************
   */

  /**
   * Overridden to provide developer readable "JsonPath" representation
   * of the context.
   */
  @Override
  public final String toString()
  {
     StringBuilder sb = new StringBuilder(64);
     switch (_type) {
     case TYPE_ROOT:
        sb.append("/");
        break;
     case TYPE_ARRAY:
        sb.append('[');
        sb.append(getCurrentIndex());
        sb.append(']');
        break;
     case TYPE_OBJECT:
        sb.append('{');
        if (_currentName != null) {
           sb.append('"');
           CharTypes.appendQuoted(sb, _currentName);
           sb.append('"');
        } else {
           sb.append('?');
        }
        sb.append('}');
        break;
     }
     return sb.toString();
  }
}
package org.codehaus.jackson;

import java.io.IOException;

/**
```

```
* Interface for objects that implement pretty printer functionality, such
* as indentation.
* Pretty printers are used to add white space in output JSON content,
* to make results more human readable. Usually this means things like adding
* linefeeds and indentation.
*/
public interface PrettyPrinter
{
  /*
  /**********************************************************
  /* First methods that act both as events, and expect
  /* output for correct functioning (i.e something gets
  /* output even when not pretty-printing)
  /**********************************************************
   */

  // // // Root-level handling:

  /**
   * Method called after a root-level value has been completely
   * output, and before another value is to be output.
   *<p>
   * Default
   * handling (without pretty-printing) will output a space, to
   * allow values to be parsed correctly. Pretty-printer is
   * to output some other suitable and nice-looking separator
   * (tab(s), space(s), linefeed(s) or any combination thereof).
   */
  public void writeRootValueSeparator(JsonGenerator jg)
     throws IOException, JsonGenerationException;

  // // Object handling

  /**
   * Method called when an Object value is to be output, before
   * any fields are output.
   *<p>
   * Default handling (without pretty-printing) will output
   * the opening curly bracket.
   * Pretty-printer is
   * to output a curly bracket as well, but can surround that
   * with other (white-space) decoration.
   */
  public void writeStartObject(JsonGenerator jg)
     throws IOException, JsonGenerationException;

  /**
   * Method called after an Object value has been completely output
```

```
     * (minus closing curly bracket).
     *<p>
     * Default handling (without pretty-printing) will output
     * the closing curly bracket.
     * Pretty-printer is
     * to output a curly bracket as well, but can surround that
     * with other (white-space) decoration.
     *
     * @param nrOfEntries Number of direct members of the array that
     *   have been output
     */
    public void writeEndObject(JsonGenerator jg, int nrOfEntries)
        throws IOException, JsonGenerationException;

    /**
     * Method called after an object entry (field:value) has been completely
     * output, and before another value is to be output.
     *<p>
     * Default handling (without pretty-printing) will output a single
     * comma to separate the two. Pretty-printer is
     * to output a comma as well, but can surround that with other
     * (white-space) decoration.
     */
    public void writeObjectEntrySeparator(JsonGenerator jg)
        throws IOException, JsonGenerationException;

    /**
     * Method called after an object field has been output, but
     * before the value is output.
     *<p>
     * Default handling (without pretty-printing) will output a single
     * colon to separate the two. Pretty-printer is
     * to output a colon as well, but can surround that with other
     * (white-space) decoration.
     */
    public void writeObjectFieldValueSeparator(JsonGenerator jg)
        throws IOException, JsonGenerationException;

    // // // Array handling

    /**
     * Method called when an Array value is to be output, before
     * any member/child values are output.
     *<p>
     * Default handling (without pretty-printing) will output
     * the opening bracket.
     * Pretty-printer is
     * to output a bracket as well, but can surround that
```

```
 * with other (white-space) decoration.
 */
public void writeStartArray(JsonGenerator jg)
   throws IOException, JsonGenerationException;

/**
 * Method called after an Array value has been completely output
 * (minus closing bracket).
 *<p>
 * Default handling (without pretty-printing) will output
 * the closing bracket.
 * Pretty-printer is
 * to output a bracket as well, but can surround that
 * with other (white-space) decoration.
 *
 * @param nrOfValues Number of direct members of the array that
 *   have been output
 */
public void writeEndArray(JsonGenerator jg, int nrOfValues)
   throws IOException, JsonGenerationException;

/**
 * Method called after an array value has been completely
 * output, and before another value is to be output.
 *<p>
 * Default handling (without pretty-printing) will output a single
 * comma to separate the two. Pretty-printer is
 * to output a comma as well, but can surround that with other
 * (white-space) decoration.
 */
public void writeArrayValueSeparator(JsonGenerator jg)
   throws IOException, JsonGenerationException;

/*
/**********************************************************
/* Then events that by default do not produce any output
/* but that are often overridden to add white space
/* in pretty-printing mode
/**********************************************************
 */

/**
 * Method called after array start marker has been output,
 * and right before the first value is to be output.
 * It is <b>not</b> called for arrays with no values.
 *<p>
 * Default handling does not output anything, but pretty-printer
 * is free to add any white space decoration.
```

```
     */
    public void beforeArrayValues(JsonGenerator jg)
        throws IOException, JsonGenerationException;

    /**
     * Method called after object start marker has been output,
     * and right before the field name of the first entry is
     * to be output.
     * It is <b>not</b> called for objects without entries.
     *<p>
     * Default handling does not output anything, but pretty-printer
     * is free to add any white space decoration.
     */
    public void beforeObjectEntries(JsonGenerator jg)
        throws IOException, JsonGenerationException;
}
package org.codehaus.jackson;

/**
* Enumeration for basic token types used for returning results
* of parsing JSON content.
*/
public enum JsonToken
{
    /* Some notes on implementation:
     *
     * - Entries are to be ordered such that start/end array/object
     *   markers come first, then field name marker (if any), and
     *   finally scalar value tokens. This is assumed by some
     *   typing checks.
     */

    /**
     * NOT_AVAILABLE can be returned if {@link JsonParser}
     * implementation can not currently return the requested
     * token (usually next one), or even if any will be
     * available, but that may be able to determine this in
     * future. This is the case with non-blocking parsers --
     * they can not block to wait for more data to parse and
     * must return something.
     *
     * @since 0.9.7
     */
    NOT_AVAILABLE(null),

    /**
     * START_OBJECT is returned when encountering '{'
     * which signals starting of an Object value.
```

```
 */
START_OBJECT("{"),

/**
 * START_OBJECT is returned when encountering '}'
 * which signals ending of an Object value
 */
END_OBJECT("}"),

/**
 * START_OBJECT is returned when encountering '['
 * which signals starting of an Array value
 */
START_ARRAY("["),

/**
 * START_OBJECT is returned when encountering ']'
 * which signals ending of an Array value
 */
END_ARRAY("]"),

/**
 * FIELD_NAME is returned when a String token is encountered
 * as a field name (same lexical value, different function)
 */
FIELD_NAME(null),

/**
 * Placeholder token returned when the input source has a concept
 * of embedded Object that are not accessible as usual structure
 * (of starting with {@link #START_OBJECT}, having values, ending with
 * {@link #END_OBJECT}), but as "raw" objects.
 *<p>
 * Note: this token is never returned by regular JSON readers, but
 * only by readers that expose other kinds of source (like
 * {@link JsonNode}-based JSON trees, Maps, Lists and such).
 *
 * @since 1.1
 */
VALUE_EMBEDDED_OBJECT(null),

/**
 * VALUE_STRING is returned when a String token is encountered
 * in value context (array element, field value, or root-level
 * stand-alone value)
 */
VALUE_STRING(null),
```

```
/**
 * VALUE_NUMBER_INT is returned when an integer numeric token is
 * encountered in value context: that is, a number that does
 * not have floating point or exponent marker in it (consists
 * only of an optional sign, followed by one or more digits)
 */
VALUE_NUMBER_INT(null),

/**
 * VALUE_NUMBER_INT is returned when a numeric token other
 * that is not an integer is encountered: that is, a number that does
 * have floating point or exponent marker in it, in addition
 * to one or more digits.
 */
VALUE_NUMBER_FLOAT(null),

/**
 * VALUE_TRUE is returned when encountering literal "true" in
 * value context
 */
VALUE_TRUE("true"),

/**
 * VALUE_FALSE is returned when encountering literal "false" in
 * value context
 */
VALUE_FALSE("false"),

/**
 * VALUE_NULL is returned when encountering literal "null" in
 * value context
 */
VALUE_NULL("null")
   ;

final String _serialized;

final char[] _serializedChars;

final byte[] _serializedBytes;

/**
 * @param Textual representation for this token, if there is a
 *   single static representation; null otherwise
 */
JsonToken(String token)
{
   if (token == null) {
```

```
            _serialized = null;
            _serializedChars = null;
            _serializedBytes = null;
        } else {
            _serialized = token;
            _serializedChars = token.toCharArray();
            // It's all in ascii, can just case...
            int len = _serializedChars.length;
            _serializedBytes = new byte[len];
            for (int i = 0; i < len; ++i) {
                _serializedBytes[i] = (byte) _serializedChars[i];
            }
        }
    }

    public String asString() { return _serialized; }
    public char[] asCharArray() { return _serializedChars; }
    public byte[] asByteArray() { return _serializedBytes; }

    public boolean isNumeric() {
        return (this == VALUE_NUMBER_INT) || (this == VALUE_NUMBER_FLOAT);
    }

    /**
     * Method that can be used to check whether this token represents
     * a valid non-structured value. This means all tokens other than
     * Object/Array start/end markers all field names.
     */
    public boolean isScalarValue() {
        // note: up to 1.5, VALUE_EMBEDDED_OBJECT was incorrectly considered non-scalar!
        return ordinal() >= VALUE_EMBEDDED_OBJECT.ordinal();
    }
}
package org.codehaus.jackson.sym;

/**
* Generic implementation of PName used for "long" names, where long
* means that its byte (UTF-8) representation is 13 bytes or more.
*/
public final class NameN
    extends Name
{
    final int[] mQuads;
    final int mQuadLen;

    NameN(String name, int hash, int[] quads, int quadLen)
    {
        super(name, hash);
```

```
        /* We have specialized implementations for shorter
         * names, so let's not allow runt instances here
         */
        if (quadLen < 3) {
            throw new IllegalArgumentException("Qlen must >= 3");
        }
        mQuads = quads;
        mQuadLen = quadLen;
    }

    // Implies quad length == 1, never matches
    @Override
    public boolean equals(int quad) { return false; }

    // Implies quad length == 2, never matches
    @Override
    public boolean equals(int quad1, int quad2) { return false; }

    @Override
    public boolean equals(int[] quads, int qlen)
    {
        if (qlen != mQuadLen) {
            return false;
        }

        /* 26-Nov-2008, tatus: Strange, but it does look like
         *   unrolling here is counter-productive, reducing
         *   speed. Perhaps it prevents inlining by HotSpot or
         *   something...
         */
        // Will always have >= 3 quads, can unroll
        /*
        if (quads[0] == mQuads[0]
            && quads[1] == mQuads[1]
            && quads[2] == mQuads[2]) {
            for (int i = 3; i < qlen; ++i) {
                if (quads[i] != mQuads[i]) {
                    return false;
                }
            }
            return true;
        }
        */

        // or simpler way without unrolling:
        for (int i = 0; i < qlen; ++i) {
            if (quads[i] != mQuads[i]) {
                return false;
```

```
        }
      }
      return true;
    }
}
package org.codehaus.jackson.sym;

/**
* Specialized implementation of PName: can be used for short Strings
* that consists of 5 to 8 bytes. Usually this means relatively short
* ascii-only names.
*<p>
* The reason for such specialized classes is mostly space efficiency;
* and to a lesser degree performance. Both are achieved for short
* Strings by avoiding another level of indirection (via quad arrays)
*/
public final class Name2
  extends Name
{
  final int mQuad1;

  final int mQuad2;

  Name2(String name, int hash, int quad1, int quad2)
  {
    super(name, hash);
    mQuad1 = quad1;
    mQuad2 = quad2;
  }

  @Override
  public boolean equals(int quad) { return false; }

  @Override
  public boolean equals(int quad1, int quad2)
  {
    return (quad1 == mQuad1) && (quad2 == mQuad2);
  }

  @Override
  public boolean equals(int[] quads, int qlen)
  {
    return (qlen == 2 && quads[0] == mQuad1 && quads[1] == mQuad2);
  }
}
package org.codehaus.jackson.sym;

/**
```

```
* Specialized implementation of PName: can be used for short Strings
* that consists of 9 to 12 bytes. It's the longest special purpose
* implementaion; longer ones are expressed using {@link NameN}.
*/
public final class Name3
  extends Name
{
  final int mQuad1;
  final int mQuad2;
  final int mQuad3;

  Name3(String name, int hash, int q1, int q2, int q3)
  {
    super(name, hash);
    mQuad1 = q1;
    mQuad2 = q2;
    mQuad3 = q3;
  }

  // Implies quad length == 1, never matches
  @Override
  public boolean equals(int quad) { return false; }

  // Implies quad length == 2, never matches
  @Override
  public boolean equals(int quad1, int quad2) { return false; }

  @Override
  public boolean equals(int[] quads, int qlen)
  {
    return (qlen == 3)
      && (quads[0] == mQuad1)
      && (quads[1] == mQuad2)
      && (quads[2] == mQuad3);
  }
}
package org.codehaus.jackson.sym;

import java.util.Arrays;

import org.codehaus.jackson.util.InternCache;

/**
* This class is basically a caching symbol table implementation used for
* canonicalizing {@link Name}s, constructed directly from a byte-based
* input source.
*
* @author Tatu Saloranta
```

```
*/
public final class BytesToNameCanonicalizer
{
  protected static final int DEFAULT_TABLE_SIZE = 64;

  /**
   * Let's not expand symbol tables past some maximum size;
   * this should protected against OOMEs caused by large documents
   * with uniquer (~= random) names.
   *
   * @since 1.5
   */
  protected static final int MAX_TABLE_SIZE = 0x10000; // 64k entries == 256k mem

  /**
   * Let's only share reasonably sized symbol tables. Max size set to 3/4 of 16k;
   * this corresponds to 64k main hash index. This should allow for enough distinct
   * names for almost any case.
   */
  final static int MAX_ENTRIES_FOR_REUSE = 6000;

  final static int MIN_HASH_SIZE = 16;

  final static int INITIAL_COLLISION_LEN = 32;

  /**
   * Bucket index is 8 bits, and value 0 is reserved to represent
   * 'empty' status.
   */
  final static int LAST_VALID_BUCKET = 0xFE;

  /*
  /**********************************************************
  /* Linkage, needed for merging symbol tables
  /**********************************************************
   */

  final BytesToNameCanonicalizer _parent;

  /*
  /**********************************************************
  /* Main table state
  /**********************************************************
   */

  /**
   * Whether canonial symbol Strings are to be intern()ed before added
   * to the table or not
```

```
 */
final boolean _intern;

// // // First, global information

/**
 * Total number of Names in the symbol table
 */
private int _count;

// // // Then information regarding primary hash array and its
// // // matching Name array

/**
 * Mask used to truncate 32-bit hash value to current hash array
 * size; essentially, hash array size - 1 (since hash array sizes
 * are 2^N).
 */
private int _mainHashMask;

/**
 * Array of 2^N size, which contains combination
 * of 24-bits of hash (0 to indicate 'empty' slot),
 * and 8-bit collision bucket index (0 to indicate empty
 * collision bucket chain; otherwise subtract one from index)
 */
private int[] _mainHash;

/**
 * Array that contains <code>Name</code> instances matching
 * entries in <code>_mainHash</code>. Contains nulls for unused
 * entries.
 */
private Name[] _mainNames;

// // // Then the collision/spill-over area info

/**
 * Array of heads of collision bucket chains; size dynamically
 */
private Bucket[] _collList;

/**
 * Total number of Names in collision buckets (included in
 * <code>_count</code> along with primary entries)
 */
private int _collCount;
```

```
/**
 * Index of the first unused collision bucket entry (== size of
 * the used portion of collision list): less than
 * or equal to 0xFF (255), since max number of entries is 255
 * (8-bit, minus 0 used as 'empty' marker)
 */
private int _collEnd;

// // // Info regarding pending rehashing...

/**
 * This flag is set if, after adding a new entry, it is deemed
 * that a rehash is warranted if any more entries are to be added.
 */
private transient boolean _needRehash;

/*
/**********************************************************
/* Sharing, versioning
/**********************************************************
 */

// // // Which of the buffers may be shared (and are copy-on-write)?

/**
 * Flag that indicates whether underlying data structures for
 * the main hash area are shared or not. If they are, then they
 * need to be handled in copy-on-write way, i.e. if they need
 * to be modified, a copy needs to be made first; at this point
 * it will not be shared any more, and can be modified.
 *<p>
 * This flag needs to be checked both when adding new main entries,
 * and when adding new collision list queues (i.e. creating a new
 * collision list head entry)
 */
private boolean _mainHashShared;

private boolean _mainNamesShared;

/**
 * Flag that indicates whether underlying data structures for
 * the collision list are shared or not. If they are, then they
 * need to be handled in copy-on-write way, i.e. if they need
 * to be modified, a copy needs to be made first; at this point
 * it will not be shared any more, and can be modified.
 *<p>
 * This flag needs to be checked when adding new collision entries.
 */
```

```
private boolean _collListShared;

/*
/**********************************************************
/* Construction, merging
/**********************************************************
 */

public static BytesToNameCanonicalizer createRoot()
{
   return new BytesToNameCanonicalizer(DEFAULT_TABLE_SIZE, true);
}

/**
 * @param intern Whether canonical symbol Strings should be interned
 *   or not
 */
public synchronized BytesToNameCanonicalizer makeChild(boolean canonicalize,
   boolean intern)
{
   return new BytesToNameCanonicalizer(this, intern);
}

/**
 * Method called by the using code to indicate it is done
 * with this instance. This lets instance merge accumulated
 * changes into parent (if need be), safely and efficiently,
 * and without calling code having to know about parent
 * information
 */
public void release()
{
   if (maybeDirty() && _parent != null) {
      _parent.mergeChild(this);
      /* Let's also mark this instance as dirty, so that just in
       * case release was too early, there's no corruption
       * of possibly shared data.
       */
      markAsShared();
   }
}

private BytesToNameCanonicalizer(int hashSize, boolean intern)
{
   _parent = null;
   _intern = intern;
   /* Sanity check: let's now allow hash sizes below certain
    * min. value
```

```
     */
    if (hashSize < MIN_HASH_SIZE) {
        hashSize = MIN_HASH_SIZE;
    } else {
        /* Also; size must be 2^N; otherwise hash algorithm won't
         * work... so let's just pad it up, if so
         */
        if ((hashSize & (hashSize - 1)) != 0) { // only true if it's 2^N
            int curr = MIN_HASH_SIZE;
            while (curr < hashSize) {
                curr += curr;
            }
            hashSize = curr;
        }
    }
    initTables(hashSize);
}

/**
 * Constructor used when creating a child instance
 */
private BytesToNameCanonicalizer(BytesToNameCanonicalizer parent, boolean intern)
{
    _parent = parent;
    _intern = intern;

    // First, let's copy the state as is:
    _count = parent._count;
    _mainHashMask = parent._mainHashMask;
    _mainHash = parent._mainHash;
    _mainNames = parent._mainNames;
    _collList = parent._collList;
    _collCount = parent._collCount;
    _collEnd = parent._collEnd;
    _needRehash = false;
    // And consider all shared, so far:
    _mainHashShared = true;
    _mainNamesShared = true;
    _collListShared = true;
}

private void initTables(int hashSize)
{
    _count = 0;
    _mainHash = new int[hashSize];
    _mainNames = new Name[hashSize];
    _mainHashShared = false;
    _mainNamesShared = false;
```

```
    _mainHashMask = hashSize - 1;

    _collListShared = true; // just since it'll need to be allocated
    _collList = null;
    _collEnd = 0;

    _needRehash = false;
}

private synchronized void mergeChild(BytesToNameCanonicalizer child)
{
    // Only makes sense if child has more entries
    int childCount = child._count;
    if (childCount <= _count) {
        return;
    }

    /* One caveat: let's try to avoid problems with
     * degenerate cases of documents with generated "random"
     * names: for these, symbol tables would bloat indefinitely.
     * One way to do this is to just purge tables if they grow
     * too large, and that's what we'll do here.
     */
    if (child.size() > MAX_ENTRIES_FOR_REUSE) {
        /* Should there be a way to get notified about this
         * event, to log it or such? (as it's somewhat abnormal
         * thing to happen)
         */
        // At any rate, need to clean up the tables, then:
        initTables(DEFAULT_TABLE_SIZE);
    } else {
        _count = child._count;
        _mainHash = child._mainHash;
        _mainNames = child._mainNames;
        _mainHashShared = true; // shouldn't matter for parent
        _mainNamesShared = true; // - "" -
        _mainHashMask = child._mainHashMask;
        _collList = child._collList;
        _collCount = child._collCount;
        _collEnd = child._collEnd;
    }
}

private void markAsShared()
{
    _mainHashShared = true;
    _mainNamesShared = true;
    _collListShared = true;
```

```
}

/*
/**********************************************************
/* API, accessors
/**********************************************************
 */

public int size() { return _count; }

/**
 * Method called to check to quickly see if a child symbol table
 * may have gotten additional entries. Used for checking to see
 * if a child table should be merged into shared table.
 */
public boolean maybeDirty()
{
   return !_mainHashShared;
}

public static Name getEmptyName()
{
   return Name1.getEmptyName();
}

/**
 * Finds and returns name matching the specified symbol, if such
 * name already exists in the table.
 * If not, will return null.
 *<p>
 * Note: separate methods to optimize common case of
 * short element/attribute names (4 or less ascii characters)
 *
 * @param firstQuad int32 containing first 4 bytes of the name;
 *   if the whole name less than 4 bytes, padded with zero bytes
 *   in front (zero MSBs, ie. right aligned)
 *
 * @return Name matching the symbol passed (or constructed for
 *   it)
 */
public Name findName(int firstQuad)
{
   int hash = calcHash(firstQuad);
   int ix = (hash & _mainHashMask);
   int val = _mainHash[ix];

   /* High 24 bits of the value are low 24 bits of hash (low 8 bits
    * are bucket index)... match?
```

```
     */
    if ((((val >> 8) ^ hash) << 8) == 0) { // match
        // Ok, but do we have an actual match?
        Name name = _mainNames[ix];
        if (name == null) { // main slot empty; can't find
            return null;
        }
        if (name.equals(firstQuad)) {
            return name;
        }
    } else if (val == 0) { // empty slot? no match
        return null;
    }
    // Maybe a spill-over?
    val &= 0xFF;
    if (val > 0) { // 0 means 'empty'
        val -= 1; // to convert from 1-based to 0...
        Bucket bucket = _collList[val];
        if (bucket != null) {
            return bucket.find(hash, firstQuad, 0);
        }
    }
    // Nope, no match whatsoever
    return null;
}

/**
 * Finds and returns name matching the specified symbol, if such
 * name already exists in the table.
 * If not, will return null.
 *<p>
 * Note: separate methods to optimize common case of relatively
 * short element/attribute names (8 or less ascii characters)
 *
 * @param firstQuad int32 containing first 4 bytes of the name.
 * @param secondQuad int32 containing bytes 5 through 8 of the
 *   name; if less than 8 bytes, padded with up to 3 zero bytes
 *   in front (zero MSBs, ie. right aligned)
 *
 * @return Name matching the symbol passed (or constructed for
 *   it)
 */
public Name findName(int firstQuad, int secondQuad)
{
    int hash = calcHash(firstQuad, secondQuad);
    int ix = (hash & _mainHashMask);
    int val = _mainHash[ix];
```

```
    /* High 24 bits of the value are low 24 bits of hash (low 8 bits
     * are bucket index)... match?
     */
    if ((((val >> 8) ^ hash) << 8) == 0) { // match
        // Ok, but do we have an actual match?
        Name name = _mainNames[ix];
        if (name == null) { // main slot empty; can't find
            return null;
        }
        if (name.equals(firstQuad, secondQuad)) {
            return name;
        }
    } else if (val == 0) { // empty slot? no match
        return null;
    }
    // Maybe a spill-over?
    val &= 0xFF;
    if (val > 0) { // 0 means 'empty'
        val -= 1; // to convert from 1-based to 0...
        Bucket bucket = _collList[val];
        if (bucket != null) {
            return bucket.find(hash, firstQuad, secondQuad);
        }
    }
    // Nope, no match whatsoever
    return null;
}

/**
 * Finds and returns name matching the specified symbol, if such
 * name already exists in the table; or if not, creates name object,
 * adds to the table, and returns it.
 *<p>
 * Note: this is the general purpose method that can be called for
 * names of any length. However, if name is less than 9 bytes long,
 * it is preferable to call the version optimized for short
 * names.
 *
 * @param quads Array of int32s, each of which contain 4 bytes of
 *   encoded name
 * @param qlen Number of int32s, starting from index 0, in quads
 *   parameter
 *
 * @return Name matching the symbol passed (or constructed for
 *   it)
 */
public Name findName(int[] quads, int qlen)
{
```

```
    /* // Not needed, never gets called
    if (qlen < 3) { // another sanity check
        return findName(quads[0], (qlen < 2) ? 0 : quads[1]);
    }
    */
    int hash = calcHash(quads, qlen);
    // (for rest of comments regarding logic, see method above)
    int ix = (hash & _mainHashMask);
    int val = _mainHash[ix];
    if ((((val >> 8) ^ hash) << 8) == 0) {
        Name name = _mainNames[ix];
        if (name == null // main slot empty; no collision list then either
            || name.equals(quads, qlen)) { // should be match, let's verify
            return name;
        }
    } else if (val == 0) { // empty slot? no match
        return null;
    }
    val &= 0xFF;
    if (val > 0) { // 0 means 'empty'
        val -= 1; // to convert from 1-based to 0...
        Bucket bucket = _collList[val];
        if (bucket != null) {
            return bucket.find(hash, quads, qlen);
        }
    }
    return null;
}

/*
/**********************************************************
/* API, mutators
/**********************************************************
 */

/**
 * @since 1.6.0
 */
public Name addName(String symbolStr, int q1, int q2)
{
    if (_intern) {
        symbolStr = InternCache.instance.intern(symbolStr);
    }
    int hash = (q2 == 0) ? calcHash(q1) : calcHash(q1, q2);
    Name symbol = constructName(hash, symbolStr, q1, q2);
    _addSymbol(hash, symbol);
    return symbol;
}
```

```
public Name addName(String symbolStr, int[] quads, int qlen)
{
   if (_intern) {
      symbolStr = InternCache.instance.intern(symbolStr);
   }
   int hash = calcHash(quads, qlen);
   Name symbol = constructName(hash, symbolStr, quads, qlen);
   _addSymbol(hash, symbol);
   return symbol;
}

/*
/**********************************************************
/* Helper methods
/**********************************************************
 */

public final static int calcHash(int firstQuad)
{
   int hash = firstQuad;
   hash ^= (hash >>> 16); // to xor hi- and low- 16-bits
   hash ^= (hash >>> 8); // as well as lowest 2 bytes
   return hash;
}

public final static int calcHash(int firstQuad, int secondQuad)
{
   int hash = (firstQuad * 31) + secondQuad;

   // If this was called for single-quad instance:
   //int hash = (secondQuad == 0) ? firstQuad : ((firstQuad * 31) + secondQuad);

   hash ^= (hash >>> 16); // to xor hi- and low- 16-bits
   hash ^= (hash >>> 8); // as well as lowest 2 bytes
   return hash;
}

public final static int calcHash(int[] quads, int qlen)
{
   // Note: may be called for qlen < 3
   int hash = quads[0];
   for (int i = 1; i < qlen; ++i) {
      hash = (hash * 31) + quads[i];
   }

   hash ^= (hash >>> 16); // to xor hi- and low- 16-bits
   hash ^= (hash >>> 8); // as well as lowest 2 bytes
```

```
    return hash;
}

/* 26-Nov-2008, tatu: not used currently; if not used in near future,
 *   let's just delete it.
 */
/*
public static int[] calcQuads(byte[] wordBytes)
{
    int blen = wordBytes.length;
    int[] result = new int[(blen + 3) / 4];
    for (int i = 0; i < blen; ++i) {
        int x = wordBytes[i] & 0xFF;

        if (++i < blen) {
            x = (x << 8) | (wordBytes[i] & 0xFF);
            if (++i < blen) {
                x = (x << 8) | (wordBytes[i] & 0xFF);
                if (++i < blen) {
                    x = (x << 8) | (wordBytes[i] & 0xFF);
                }
            }
        }
        result[i >> 2] = x;
    }
    return result;
}
*/

/*
/**********************************************************
/* Standard methods
/**********************************************************
 */

/*
@Override
public String toString()
{
    StringBuilder sb = new StringBuilder();
    sb.append("[BytesToNameCanonicalizer, size: ");
    sb.append(_count);
    sb.append('/');
    sb.append(_mainHash.length);
    sb.append(", ");
    sb.append(_collCount);
    sb.append(" coll; avg length: ");
```

```
    // Average length: minimum of 1 for all (1 == primary hit);
    // and then 1 per each traversal for collisions/buckets
    //int maxDist = 1;
    int pathCount = _count;
    for (int i = 0; i < _collEnd; ++i) {
        int spillLen = _collList[i].length();
        for (int j = 1; j <= spillLen; ++j) {
            pathCount += j;
        }
    }
    double avgLength;

    if (_count == 0) {
        avgLength = 0.0;
    } else {
        avgLength = (double) pathCount / (double) _count;
    }
    // let's round up a bit (two 2 decimal places)
    //avgLength -= (avgLength % 0.01);

    sb.append(avgLength);
    sb.append(']');
    return sb.toString();
}
*/

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

private void _addSymbol(int hash, Name symbol)
{
    if (_mainHashShared) { // always have to modify main entry
        unshareMain();
    }
    // First, do we need to rehash?
    if (_needRehash) {
        rehash();
    }

    ++_count;

    /* Ok, enough about set up: now we need to find the slot to add
     * symbol in:
     */
```

```
int ix = (hash & _mainHashMask);
if (_mainNames[ix] == null) { // primary empty?
    _mainHash[ix] = (hash << 8);
    if (_mainNamesShared) {
        unshareNames();
    }
    _mainNames[ix] = symbol;
} else { // nope, it's a collision, need to spill over
    /* How about spill-over area... do we already know the bucket
     * (is the case if it's not the first collision)
     */
    if (_collListShared) {
        unshareCollision(); // also allocates if list was null
    }

    ++_collCount;
    int entryValue = _mainHash[ix];
    int bucket = entryValue & 0xFF;
    if (bucket == 0) { // first spill over?
        if (_collEnd <= LAST_VALID_BUCKET) { // yup, still unshared bucket
            bucket = _collEnd;
            ++_collEnd;
            // need to expand?
            if (bucket >= _collList.length) {
                expandCollision();
            }
        } else { // nope, have to share... let's find shortest?
            bucket = findBestBucket();
        }
        // Need to mark the entry... and the spill index is 1-based
        _mainHash[ix] = (entryValue & ~0xFF) | (bucket + 1);
    } else {
        --bucket; // 1-based index in value
    }

    // And then just need to link the new bucket entry in
    _collList[bucket] = new Bucket(symbol, _collList[bucket]);
}

/* Ok. Now, do we need a rehash next time? Need to have at least
 * 50% fill rate no matter what:
 */
{
    int hashSize = _mainHash.length;
    if (_count > (hashSize >> 1)) {
        int hashQuarter = (hashSize >> 2);
        /* And either strictly above 75% (the usual) or
         * just 50%, and collision count >= 25% of total hash size
```

```
             */
            if (_count > (hashSize - hashQuarter)) {
                _needRehash = true;
            } else if (_collCount >= hashQuarter) {
                _needRehash = true;
            }
        }
    }
}

private void rehash()
{
    _needRehash = false;
    // Note: since we'll make copies, no need to unshare, can just mark as such:
    _mainNamesShared = false;

    /* And then we can first deal with the main hash area. Since we
     * are expanding linearly (double up), we know there'll be no
     * collisions during this phase.
     */
    int[] oldMainHash = _mainHash;
    int len = oldMainHash.length;
    int newLen = len+len;

    /* 13-Mar-2010, tatu: Let's guard against OOME that could be caused by
     *    large documents with unique (or mostly so) names
     */
    if (newLen > MAX_TABLE_SIZE) {
        nukeSymbols();
        return;
    }

    _mainHash = new int[newLen];
    _mainHashMask = (newLen - 1);
    Name[] oldNames = _mainNames;
    _mainNames = new Name[newLen];
    int symbolsSeen = 0; // let's do a sanity check
    for (int i = 0; i < len; ++i) {
        Name symbol = oldNames[i];
        if (symbol != null) {
            ++symbolsSeen;
            int hash = symbol.hashCode();
            int ix = (hash & _mainHashMask);
            _mainNames[ix] = symbol;
            _mainHash[ix] = hash << 8; // will clear spill index
        }
    }
```

```
/* And then the spill area. This may cause collisions, although
 * not necessarily as many as there were earlier. Let's allocate
 * same amount of space, however
 */
int oldEnd = _collEnd;
if (oldEnd == 0) { // no prior collisions...
   return;
}

_collCount = 0;
_collEnd = 0;
_collListShared = false;

Bucket[] oldBuckets = _collList;
_collList = new Bucket[oldBuckets.length];
for (int i = 0; i < oldEnd; ++i) {
   for (Bucket curr = oldBuckets[i]; curr != null; curr = curr._next) {
      ++symbolsSeen;
      Name symbol = curr._name;
      int hash = symbol.hashCode();
      int ix = (hash & _mainHashMask);
      int val = _mainHash[ix];
      if (_mainNames[ix] == null) { // no primary entry?
         _mainHash[ix] = (hash << 8);
         _mainNames[ix] = symbol;
      } else { // nope, it's a collision, need to spill over
         ++_collCount;
         int bucket = val & 0xFF;
         if (bucket == 0) { // first spill over?
            if (_collEnd <= LAST_VALID_BUCKET) { // yup, still unshared bucket
               bucket = _collEnd;
               ++_collEnd;
               // need to expand?
               if (bucket >= _collList.length) {
                  expandCollision();
               }
            } else { // nope, have to share... let's find shortest?
               bucket = findBestBucket();
            }
            // Need to mark the entry... and the spill index is 1-based
            _mainHash[ix] = (val & ~0xFF) | (bucket + 1);
         } else {
            --bucket; // 1-based index in value
         }
         // And then just need to link the new bucket entry in
         _collList[bucket] = new Bucket(symbol, _collList[bucket]);
      }
   } // for (... buckets in the chain ...)
```

```
    } // for (... list of bucket heads ... )

    if (symbolsSeen != _count) { // sanity check
        throw new RuntimeException("Internal error: count after rehash "+symbolsSeen+"; should be "+_count);
    }
}

/**
 * Helper method called to empty all shared symbols, but to leave
 * arrays allocated
 */
private void nukeSymbols()
{
    _count = 0;
    Arrays.fill(_mainHash, 0);
    Arrays.fill(_mainNames, null);
    Arrays.fill(_collList, null);
    _collCount = 0;
    _collEnd = 0;
}

/**
 * Method called to find the best bucket to spill a Name over to:
 * usually the first bucket that has only one entry, but in general
 * first one of the buckets with least number of entries
 */
private int findBestBucket()
{
    Bucket[] buckets = _collList;
    int bestCount = Integer.MAX_VALUE;
    int bestIx = -1;

    for (int i = 0, len = _collEnd; i < len; ++i) {
        int count = buckets[i].length();
        if (count < bestCount) {
            if (count == 1) { // best possible
                return i;
            }
            bestCount = count;
            bestIx = i;
        }
    }
    return bestIx;
}

/**
 * Method that needs to be called, if the main hash structure
 * is (may be) shared. This happens every time something is added,
```

```
 * even if addition is to the collision list (since collision list
 * index comes from lowest 8 bits of the primary hash entry)
 */
private void unshareMain()
{
    int[] old = _mainHash;
    int len = _mainHash.length;

    _mainHash = new int[len];
    System.arraycopy(old, 0, _mainHash, 0, len);
    _mainHashShared = false;
}

private void unshareCollision()
{
    Bucket[] old = _collList;
    if (old == null) {
        _collList = new Bucket[INITIAL_COLLISION_LEN];
    } else {
        int len = old.length;
        _collList = new Bucket[len];
        System.arraycopy(old, 0, _collList, 0, len);
    }
    _collListShared = false;
}

private void unshareNames()
{
    Name[] old = _mainNames;
    int len = old.length;
    _mainNames = new Name[len];
    System.arraycopy(old, 0, _mainNames, 0, len);
    _mainNamesShared = false;
}

private void expandCollision()
{
    Bucket[] old = _collList;
    int len = old.length;
    _collList = new Bucket[len+len];
    System.arraycopy(old, 0, _collList, 0, len);
}


/*
/**********************************************************
/* Constructing name objects
/**********************************************************
```

```
 */

private static Name constructName(int hash, String name, int q1, int q2)
{
   if (q2 == 0) { // one quad only?
      return new Name1(name, hash, q1);
   }
   return new Name2(name, hash, q1, q2);
}

private static Name constructName(int hash, String name, int[] quads, int qlen)
{
   if (qlen < 4) { // Need to check for 3 quad one, can do others too
      switch (qlen) {
      case 1:
         return new Name1(name, hash, quads[0]);
      case 2:
         return new Name2(name, hash, quads[0], quads[1]);
      case 3:
         return new Name3(name, hash, quads[0], quads[1], quads[2]);
      default:
      }
   }
   // Otherwise, need to copy the incoming buffer
   int[] buf = new int[qlen];
   for (int i = 0; i < qlen; ++i) {
      buf[i] = quads[i];
   }
   return new NameN(name, hash, buf, qlen);
}

/*
/************************************************************
/* Helper classes
/************************************************************
 */

final static class Bucket
{
   protected final Name _name;
   protected final Bucket _next;

   Bucket(Name name, Bucket next)
   {
      _name = name;
      _next = next;
   }
```

```
    public int length()
    {
        int len = 1;
        for (Bucket curr = _next; curr != null; curr = curr._next) {
            ++len;
        }
        return len;
    }

    public Name find(int hash, int firstQuad, int secondQuad)
    {
        if (_name.hashCode() == hash) {
            if (_name.equals(firstQuad, secondQuad)) {
                return _name;
            }
        }
        for (Bucket curr = _next; curr != null; curr = curr._next) {
            Name currName = curr._name;
            if (currName.hashCode() == hash) {
                if (currName.equals(firstQuad, secondQuad)) {
                    return currName;
                }
            }
        }
        return null;
    }

    public Name find(int hash, int[] quads, int qlen)
    {
        if (_name.hashCode() == hash) {
            if (_name.equals(quads, qlen)) {
                return _name;
            }
        }
        for (Bucket curr = _next; curr != null; curr = curr._next) {
            Name currName = curr._name;
            if (currName.hashCode() == hash) {
                if (currName.equals(quads, qlen)) {
                    return currName;
                }
            }
        }
        return null;
    }
  }
}
package org.codehaus.jackson.sym;
```

```
/**
* Base class for tokenized names (key strings in objects) that have
* been tokenized from byte-based input sources (like
* {@link java.io.InputStream}.
*
* @author Tatu Saloranta
*/
public abstract class Name
{
  protected final String _name;

  protected final int _hashCode;

  protected Name(String name, int hashCode) {
     _name = name;
     _hashCode = hashCode;
  }

  public String getName() { return _name; }

  /*
  /**********************************************************
  /* Methods for package/core parser
  /**********************************************************
   */

  public abstract boolean equals(int quad1);

  public abstract boolean equals(int quad1, int quad2);

  public abstract boolean equals(int[] quads, int qlen);

  /*
  /**********************************************************
  /* Overridden standard methods
  /**********************************************************
   */

  @Override public String toString() { return _name; }

  @Override public final int hashCode() { return _hashCode; }

  @Override public boolean equals(Object o)
  {
     // Canonical instances, can usually just do identity comparison
     return (o == this);
  }
}
```

```
package org.codehaus.jackson.sym;

import java.util.Arrays;

import org.codehaus.jackson.util.InternCache;

/**
* This class is a kind of specialized type-safe Map, from char array to
* String value. Specialization means that in addition to type-safety
* and specific access patterns (key char array, Value optionally interned
* String; values added on access if necessary), and that instances are
* meant to be used concurrently, but by using well-defined mechanisms
* to obtain such concurrently usable instances. Main use for the class
* is to store symbol table information for things like compilers and
* parsers; especially when number of symbols (keywords) is limited.
*<p>
* For optimal performance, usage pattern should be one where matches
* should be very common (esp. after "warm-up"), and as with most hash-based
* maps/sets, that hash codes are uniformly distributed. Also, collisions
* are slightly more expensive than with HashMap or HashSet, since hash codes
* are not used in resolving collisions; that is, equals() comparison is
* done with all symbols in same bucket index.<br />
* Finally, rehashing is also more expensive, as hash codes are not
* stored; rehashing requires all entries' hash codes to be recalculated.
* Reason for not storing hash codes is reduced memory usage, hoping
* for better memory locality.
*<p>
* Usual usage pattern is to create a single "master" instance, and either
* use that instance in sequential fashion, or to create derived "child"
* instances, which after use, are asked to return possible symbol additions
* to master instance. In either case benefit is that symbol table gets
* initialized so that further uses are more efficient, as eventually all
* symbols needed will already be in symbol table. At that point no more
* Symbol String allocations are needed, nor changes to symbol table itself.
*<p>
* Note that while individual SymbolTable instances are NOT thread-safe
* (much like generic collection classes), concurrently used "child"
* instances can be freely used without synchronization. However, using
* master table concurrently with child instances can only be done if
* access to master instance is read-only (ie. no modifications done).
*/

public final class CharsToNameCanonicalizer
{
  /**
   * Default initial table size. Shouldn't be miniscule (as there's
   * cost to both array realloc and rehashing), but let's keep
   * it reasonably small nonetheless. For systems that properly
```

```
     * reuse factories it doesn't matter either way; but when
     * recreating factories often, initial overhead may dominate.
     */
    protected static final int DEFAULT_TABLE_SIZE = 64;

    /**
     * Let's not expand symbol tables past some maximum size;
     * this should protected against OOMEs caused by large documents
     * with uniquer (~= random) names.
     *
     * @since 1.5
     */
    protected static final int MAX_TABLE_SIZE = 0x10000; // 64k entries == 256k mem

    /**
     * Let's only share reasonably sized symbol tables. Max size set to 3/4 of 16k;
     * this corresponds to 64k main hash index. This should allow for enough distinct
     * names for almost any case.
     */
    final static int MAX_ENTRIES_FOR_REUSE = 12000;

    final static CharsToNameCanonicalizer sBootstrapSymbolTable;
    static {
       sBootstrapSymbolTable = new CharsToNameCanonicalizer();
    }

    /*
    /**********************************************
    /* Configuration:
    /**********************************************
     */

    /**
     * Sharing of learnt symbols is done by optional linking of symbol
     * table instances with their parents. When parent linkage is
     * defined, and child instance is released (call to <code>release</code>),
     * parent's shared tables may be updated from the child instance.
     */
    protected CharsToNameCanonicalizer _parent;

    /**
     * Whether canonical symbol Strings are to be intern()ed before added
     * to the table or not
     */
    final protected boolean _intern;

    /**
     * Whether any canonicalization should be attempted (whether using
```

```
 * intern or not)
 */
final protected boolean _canonicalize;

/*
/**********************************************
/* Actual symbol table data:
/**********************************************
 */

/**
 * Primary matching symbols; it's expected most match occur from
 * here.
 */
protected String[] _symbols;

/**
 * Overflow buckets; if primary doesn't match, lookup is done
 * from here.
 *<p>
 * Note: Number of buckets is half of number of symbol entries, on
 * assumption there's less need for buckets.
 */
protected Bucket[] _buckets;

/**
 * Current size (number of entries); needed to know if and when
 * rehash.
 */
protected int _size;

/**
 * Limit that indicates maximum size this instance can hold before
 * it needs to be expanded and rehashed. Calculated using fill
 * factor passed in to constructor.
 */
protected int _sizeThreshold;

/**
 * Mask used to get index from hash values; equal to
 * <code>_buckets.length - 1</code>, when _buckets.length is
 * a power of two.
 */
protected int _indexMask;

/*
/**********************************************
/* State regarding shared arrays
```

```
/**********************************************
 */

/**
 * Flag that indicates if any changes have been made to the data;
 * used to both determine if bucket array needs to be copied when
 * (first) change is made, and potentially if updated bucket list
 * is to be resync'ed back to master instance.
 */
protected boolean _dirty;

/*
/**********************************************
/* Life-cycle
/**********************************************
 */

/**
 * Method called to create root canonicalizer for a {@link org.codehaus.jackson.JsonFactory}
 * instance. Root instance is never used directly; its main use is for
 * storing and sharing underlying symbol arrays as needed.
 */
public static CharsToNameCanonicalizer createRoot()
{
   return sBootstrapSymbolTable.makeOrphan();
}

/**
 * Main method for constructing a master symbol table instance.
 *
 * @param initialSize Minimum initial size for bucket array; internally
 *   will always use a power of two equal to or bigger than this value.
 */
private CharsToNameCanonicalizer()
{
   // these settings don't really matter for the bootstrap instance
   _canonicalize = true;
   _intern = true;
   // And we'll also set flags so no copying of buckets is needed:
   _dirty = true;
   initTables(DEFAULT_TABLE_SIZE);
}

private void initTables(int initialSize)
{
   _symbols = new String[initialSize];
   _buckets = new Bucket[initialSize >> 1];
   // Mask is easy to calc for powers of two.
```

```
    _indexMask = initialSize - 1;
    _size = 0;
    // Hard-coded fill factor is 75%
    _sizeThreshold = (initialSize - (initialSize >> 2));
}

/**
 * Internal constructor used when creating child instances.
 */
private CharsToNameCanonicalizer(CharsToNameCanonicalizer parent,
        boolean canonicalize, boolean intern,
        String[] symbols, Bucket[] buckets, int size)
{
    _parent = parent;
    _canonicalize = canonicalize;
    _intern = intern;

    _symbols = symbols;
    _buckets = buckets;
    _size = size;
    // Hard-coded fill factor, 75%
    int arrayLen = (symbols.length);
    _sizeThreshold = arrayLen - (arrayLen >> 2);
    _indexMask =  (arrayLen - 1);

    // Need to make copies of arrays, if/when adding new entries
    _dirty = false;
}

/**
 * "Factory" method; will create a new child instance of this symbol
 * table. It will be a copy-on-write instance, ie. it will only use
 * read-only copy of parent's data, but when changes are needed, a
 * copy will be created.
 *<p>
 * Note: while this method is synchronized, it is generally not
 * safe to both use makeChild/mergeChild, AND to use instance
 * actively. Instead, a separate 'root' instance should be used
 * on which only makeChild/mergeChild are called, but instance itself
 * is not used as a symbol table.
 */
public synchronized CharsToNameCanonicalizer makeChild(boolean canonicalize, boolean intern)
{
    return new CharsToNameCanonicalizer(this, canonicalize, intern, _symbols, _buckets, _size);
}

private CharsToNameCanonicalizer makeOrphan()
{
```

```
    return new CharsToNameCanonicalizer(null, true, true, _symbols, _buckets, _size);
}

/**
 * Method that allows contents of child table to potentially be
 * "merged in" with contents of this symbol table.
 *<p>
 * Note that caller has to make sure symbol table passed in is
 * really a child or sibling of this symbol table.
 */
private synchronized void mergeChild(CharsToNameCanonicalizer child)
{
    /* One caveat: let's try to avoid problems with
     * degenerate cases of documents with generated "random"
     * names: for these, symbol tables would bloat indefinitely.
     * One way to do this is to just purge tables if they grow
     * too large, and that's what we'll do here.
     */
    if (child.size() > MAX_ENTRIES_FOR_REUSE) {
        /* Should there be a way to get notified about this
         * event, to log it or such? (as it's somewhat abnormal
         * thing to happen)
         */
        // At any rate, need to clean up the tables, then:
        initTables(DEFAULT_TABLE_SIZE);
    } else {
        /* Otherwise, we'll merge changed stuff in, if there are
         * more entries (which may not be the case if one of siblings
         * has added symbols first or such)
         */
        if (child.size() <= size()) { // nothing to add
            return;
        }
        // Okie dokie, let's get the data in!
        _symbols = child._symbols;
        _buckets = child._buckets;
        _size = child._size;
        _sizeThreshold = child._sizeThreshold;
        _indexMask = child._indexMask;
    }
    /* Dirty flag... well, let's just clear it, to force copying just
     * in case. Shouldn't really matter, for master tables.
     * (which this is, given something is merged to it etc)
     */
    _dirty = false;
}

public void release()
```

```
{
    // If nothing has been added, nothing to do
    if (!maybeDirty()) {
        return;
    }
    if (_parent != null) {
        _parent.mergeChild(this);
        /* Let's also mark this instance as dirty, so that just in
         * case release was too early, there's no corruption
         * of possibly shared data.
         */
        _dirty = false;
    }
}

/*
/**********************************************
/* Public API, generic accessors:
/**********************************************
 */

public int size() { return _size; }

public boolean maybeDirty() { return _dirty; }

/*
/**********************************************
/* Public API, accessing symbols:
/**********************************************
 */

public String findSymbol(char[] buffer, int start, int len, int hash)
{
    if (len < 1) { // empty Strings are simplest to handle up front
        return "";
    }
    if (!_canonicalize) { // [JACKSON-259]
        return new String(buffer, start, len);
    }

    hash &= _indexMask;

    String sym = _symbols[hash];

    // Optimal case; checking existing primary symbol for hash index:
    if (sym != null) {
        // Let's inline primary String equality checking:
        if (sym.length() == len) {
```

```
            int i = 0;
            do {
                if (sym.charAt(i) != buffer[start+i]) {
                    break;
                }
            } while (++i < len);
            // Optimal case; primary match found
            if (i == len) {
                return sym;
            }
        }
        // How about collision bucket?
        Bucket b = _buckets[hash >> 1];
        if (b != null) {
            sym = b.find(buffer, start, len);
            if (sym != null) {
                return sym;
            }
        }
    }

    if (!_dirty) { //need to do copy-on-write?
        copyArrays();
        _dirty = true;
    } else if (_size >= _sizeThreshold) { // Need to expand?
        rehash();
        /* Need to recalc hash; rare occurence (index mask has been
         * recalculated as part of rehash)
         */
        hash = calcHash(buffer, start, len) & _indexMask;
    }
    ++_size;

    String newSymbol = new String(buffer, start, len);
    if (_intern) {
        newSymbol = InternCache.instance.intern(newSymbol);
    }
    // Ok; do we need to add primary entry, or a bucket?
    if (_symbols[hash] == null) {
        _symbols[hash] = newSymbol;
    } else {
        int bix = hash >> 1;
        _buckets[bix] = new Bucket(newSymbol, _buckets[bix]);
    }

    return newSymbol;
}
```

```
/**
 * Implementation of a hashing method for variable length
 * Strings. Most of the time intention is that this calculation
 * is done by caller during parsing, not here; however, sometimes
 * it needs to be done for parsed "String" too.
 *
 * @param len Length of String; has to be at least 1 (caller guarantees
 *   this pre-condition)
 */
public static int calcHash(char[] buffer, int start, int len) {
   int hash = (int) buffer[0];
   for (int i = 1; i < len; ++i) {
      hash = (hash * 31) + (int) buffer[i];
   }
   return hash;
}

public static int calcHash(String key) {
   int hash = (int) key.charAt(0);
   for (int i = 1, len = key.length(); i < len; ++i) {
      hash = (hash * 31) + (int) key.charAt(i);

   }
   return hash;
}

/*
/**********************************************
/* Internal methods
/**********************************************
 */

/**
 * Method called when copy-on-write is needed; generally when first
 * change is made to a derived symbol table.
 */
private void copyArrays() {
   String[] oldSyms = _symbols;
   int size = oldSyms.length;
   _symbols = new String[size];
   System.arraycopy(oldSyms, 0, _symbols, 0, size);
   Bucket[] oldBuckets = _buckets;
   size = oldBuckets.length;
   _buckets = new Bucket[size];
   System.arraycopy(oldBuckets, 0, _buckets, 0, size);
}

/**
```

```
 * Method called when size (number of entries) of symbol table grows
 * so big that load factor is exceeded. Since size has to remain
 * power of two, arrays will then always be doubled. Main work
 * is really redistributing old entries into new String/Bucket
 * entries.
 */
private void rehash()
{
    int size = _symbols.length;
    int newSize = size + size;

    /* 12-Mar-2010, tatu: Let's actually limit maximum size we are
     *    prepared to use, to guard against OOME in case of unbounded
     *    name sets (unique [non-repeating] names)
     */
    if (newSize > MAX_TABLE_SIZE) {
        /* If this happens, there's no point in either growing or
         * shrinking hash areas. Rather, it's better to just clean
         * them up for reuse.
         */
        _size = 0;
        Arrays.fill(_symbols, null);
        Arrays.fill(_buckets, null);
        _dirty = true;
        return;
    }

    String[] oldSyms = _symbols;
    Bucket[] oldBuckets = _buckets;
    _symbols = new String[newSize];
    _buckets = new Bucket[newSize >> 1];
    // Let's update index mask, threshold, now (needed for rehashing)
    _indexMask = newSize - 1;
    _sizeThreshold += _sizeThreshold;

    int count = 0; // let's do sanity check

    /* Need to do two loops, unfortunately, since spill-over area is
     * only half the size:
     */
    for (int i = 0; i < size; ++i) {
        String symbol = oldSyms[i];
        if (symbol != null) {
            ++count;
            int index = calcHash(symbol) & _indexMask;
            if (_symbols[index] == null) {
                _symbols[index] = symbol;
            } else {
```

```
                int bix = index >> 1;
                _buckets[bix] = new Bucket(symbol, _buckets[bix]);
            }
        }
    }

    size >>= 1;
    for (int i = 0; i < size; ++i) {
        Bucket b = oldBuckets[i];
        while (b != null) {
            ++count;
            String symbol = b.getSymbol();
            int index = calcHash(symbol) & _indexMask;
            if (_symbols[index] == null) {
                _symbols[index] = symbol;
            } else {
                int bix = index >> 1;
                _buckets[bix] = new Bucket(symbol, _buckets[bix]);
            }
            b = b.getNext();
        }
    }

    if (count != _size) {
        throw new Error("Internal error on SymbolTable.rehash(): had "+_size+" entries; now have "+count+".");
    }
}

/*
/**********************************************
/* Bucket class
/**********************************************
 */

/**
 * This class is a symbol table entry. Each entry acts as a node
 * in a linked list.
 */
static final class Bucket {
    private final String _symbol;
    private final Bucket mNext;

    public Bucket(String symbol, Bucket next) {
        _symbol = symbol;
        mNext = next;
    }

    public String getSymbol() { return _symbol; }
```

```
    public Bucket getNext() { return mNext; }

    public String find(char[] buf, int start, int len) {
        String sym = _symbol;
        Bucket b = mNext;

        while (true) { // Inlined equality comparison:
            if (sym.length() == len) {
                int i = 0;
                do {
                    if (sym.charAt(i) != buf[start+i]) {
                        break;
                    }
                } while (++i < len);
                if (i == len) {
                    return sym;
                }
            }
            if (b == null) {
                break;
            }
            sym = b.getSymbol();
            b = b.getNext();
        }
        return null;
    }

/* 26-Nov-2008, tatu: not used currently; if not used in near future,
 *   let's just delete it.
 */
    /*
    public String find(String str) {
        String sym = _symbol;
        Bucket b = mNext;

        while (true) {
            if (sym.equals(str)) {
                return sym;
            }
            if (b == null) {
                break;
            }
            sym = b.getSymbol();
            b = b.getNext();
        }
        return null;
    }
    */
```

```
  }
}
/**
* Internal implementation classes for efficient handling of
* of symbols in JSON (field names in Objects)
*/
package org.codehaus.jackson.sym;
package org.codehaus.jackson.sym;

/**
* Specialized implementation of PName: can be used for short Strings
* that consists of at most 4 bytes. Usually this means short
* ascii-only names.
*<p>
* The reason for such specialized classes is mostly space efficiency;
* and to a lesser degree performance. Both are achieved for short
* Strings by avoiding another level of indirection (via quad arrays)
*/
public final class Name1
  extends Name
{
  final static Name1 sEmptyName = new Name1("", 0, 0);

  final int mQuad;

  Name1(String name, int hash, int quad)
  {
    super(name, hash);
    mQuad = quad;
  }

  final static Name1 getEmptyName() { return sEmptyName; }

  @Override
  public boolean equals(int quad)
  {
    return (quad == mQuad);
  }

  @Override
  public boolean equals(int quad1, int quad2)
  {
    return (quad1 == mQuad) && (quad2 == 0);
  }

  @Override
  public boolean equals(int[] quads, int qlen)
  {
```

```
        return (qlen == 1 && quads[0] == mQuad);
    }
}
package org.codehaus.jackson;

/**
* Interface that defines how Jackson package can interact with efficient
* pre-serialized or lazily-serialized and reused String representations.
* Typically implementations store possible serialized version(s) so that
* serialization of String can be done more efficiently, especially when
* used multiple times.
*
* @since 1.7 (1.6 introduced implementation, but interface extracted later)
*
* @see org.codehaus.jackson.io.SerializedString
*/
public interface SerializableString
{
    /**
     * Returns unquoted String that this object represents (and offers
     * serialized forms for)
     */
    public String getValue();

    /**
     * Returns length of the (unquoted) String as characters.
     * Functionally equvalent to:
     *<pre>
     *   getValue().length();
     *</pre>
     */
    public int charLength();

    /**
     * Returns JSON quoted form of the String, as character array. Result
     * can be embedded as-is in textual JSON as property name or JSON String.
     */
    public char[] asQuotedChars();

    /**
     * Returns UTF-8 encoded version of unquoted String.
     * Functionally equivalent to (but more efficient than):
     *<pre>
     * getValue().getBytes("UTF-8");
     *</pre>
     */
    public byte[] asUnquotedUTF8();
```

```
  /**
   * Returns UTF-8 encoded version of JSON-quoted String.
   * Functionally equivalent to (but more efficient than):
   *<pre>
   * new String(asQuotedChars()).getBytes("UTF-8");
   *</pre>
   */
  public byte[] asQuotedUTF8();

}
/* Jackson JSON-processor.
*
* Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi
*
* Licensed under the License specified in file LICENSE, included with
* the source code and binary code bundles.
* You may not use this file except in compliance with the License.
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package org.codehaus.jackson;

import java.io.*;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.type.TypeReference;

/**
* Base class that defines public API for reading JSON content.
* Instances are created using factory methods of
* a {@link JsonFactory} instance.
*
* @author Tatu Saloranta
*/
public abstract class JsonParser
  implements Closeable, Versioned
{
  private final static int MIN_BYTE_I = (int) Byte.MIN_VALUE;
  private final static int MAX_BYTE_I = (int) Byte.MAX_VALUE;

  private final static int MIN_SHORT_I = (int) Short.MIN_VALUE;
  private final static int MAX_SHORT_I = (int) Short.MAX_VALUE;
```

```
/**
 * Enumeration of possible "native" (optimal) types that can be
 * used for numbers.
 */
public enum NumberType {
   INT, LONG, BIG_INTEGER, FLOAT, DOUBLE, BIG_DECIMAL
};

/**
 * Enumeration that defines all togglable features for parsers.
 */
public enum Feature {
   /**
    * Feature that determines whether parser will automatically
    * close underlying input source that is NOT owned by the
    * parser. If disabled, calling application has to separately
    * close the underlying {@link InputStream} and {@link Reader}
    * instances used to create the parser. If enabled, parser
    * will handle closing, as long as parser itself gets closed:
    * this happens when end-of-input is encountered, or parser
    * is closed by a call to {@link JsonParser#close}.
    *<p>
    * Feature is enabled by default.
    */
   AUTO_CLOSE_SOURCE(true),

   /**
    * Feature that determines whether parser will allow use
    * of Java/C++ style comments (both '/'+'*' and
    * '//' varieties) within parsed content or not.
    *<p>
    * Since JSON specification does not mention comments as legal
    * construct,
    * this is a non-standard feature; however, in the wild
    * this is extensively used. As such, feature is
    * <b>disabled by default</b> for parsers and must be
    * explicitly enabled (via factory or parser instance).
    *<p>
    * This feature can be changed for parser instances.
    */
   ALLOW_COMMENTS(false),

   /**
    * Feature that determines whether parser will allow use
    * of unquoted field names (which is allowed by Javascript,
    * but not by JSON specification).
    *<p>
```

```
     * Since JSON specification requires use of double quotes for
     * field names,
     * this is a non-standard feature, and as such disabled by
     * default.
     *<p>
     * This feature can be changed for parser instances.
     *
     * @since 1.2
     */
    ALLOW_UNQUOTED_FIELD_NAMES(false),

    /**
     * Feature that determines whether parser will allow use
     * of single quotes (apostrophe, character '\'') for
     * quoting Strings (names and String values). If so,
     * this is in addition to other acceptabl markers.
     * but not by JSON specification).
     *<p>
     * Since JSON specification requires use of double quotes for
     * field names,
     * this is a non-standard feature, and as such disabled by
     * default.
     *<p>
     * This feature can be changed for parser instances.
     *
     * @since 1.3
     */
    ALLOW_SINGLE_QUOTES(false),

    /**
     * Feature that determines whether parser will allow
     * JSON Strings to contain unquoted control characters
     * (ASCII characters with value less than 32, including
     * tab and line feed characters) or not.
     * If feature is set false, an exception is thrown if such a
     * character is encountered.
     *<p>
     * Since JSON specification requires quoting for all control characters,
     * this is a non-standard feature, and as such disabled by default.
     *<p>
     * This feature can be changed for parser instances.
     *
     * @since 1.4
     */
    ALLOW_UNQUOTED_CONTROL_CHARS(false),

    /**
     * Feature that can be enabled to accept quoting of all character
```

```
     * using backslash qooting mechanism: if not enabled, only characters
     * that are explicitly listed by JSON specification can be thus
     * escaped (see JSON spec for small list of these characters)
     *<p>
     * Since JSON specification requires quoting for all control characters,
     * this is a non-standard feature, and as such disabled by default.
     *<p>
     * This feature can be changed for parser instances.
     *
     * @since 1.6
     */
    ALLOW_BACKSLASH_ESCAPING_ANY_CHARACTER(false),

    /**
     * Feature that determines whether JSON object field names are
     * to be canonicalized using {@link String#intern} or not:
     * if enabled, all field names will be intern()ed (and caller
     * can count on this being true for all such names); if disabled,
     * no intern()ing is done. There may still be basic
     * canonicalization (that is, same String will be used to represent
     * all identical object property names for a single document).
     *<p>
     * Note: this setting only has effect if
     * {@link #CANONICALIZE_FIELD_NAMES} is true -- otherwise no
     * canonicalization of any sort is done.
     *
     * @since 1.3
     */
    INTERN_FIELD_NAMES(true),

    /**
     * Feature that determines whether JSON object field names are
     * to be canonicalized (details of how canonicalization is done
     * then further specified by
     * {@link #INTERN_FIELD_NAMES}).
     *
     * @since 1.5
     */
    CANONICALIZE_FIELD_NAMES(true),

       // 14-Sep-2009, Tatu: This would be [JACKSON-142] implementation:
    /*
     * Feature that allows parser to recognize set of
     * "Not-a-Number" (NaN) tokens as legal floating number
     * values (similar to how many other data formats and
     * programming language source code allows it).
     * Specific subset contains values that
     * <a href="http://www.w3.org/TR/xmlschema-2/">XML Schema</a>
```

```
     * (see section 3.2.4.1, Lexical Representation)
     * allows (tokens are quoted contents, not including quotes):
     *<ul>
     *  <li>"INF" (for positive infinity)
     *  <li>"-INF" (for negative infinity)
     *  <li>"NaN" (for other not-a-numbers, like result of division by zero)
     *</ul>

     ,ALLOW_NON_NUMERIC_NUMBERS(false),
     */

        ;

    final boolean _defaultState;

    /**
     * Method that calculates bit set (flags) of all features that
     * are enabled by default.
     */
    public static int collectDefaults()
    {
        int flags = 0;
        for (Feature f : values()) {
            if (f.enabledByDefault()) {
                flags |= f.getMask();
            }
        }
        return flags;
    }

    private Feature(boolean defaultState) {
        _defaultState = defaultState;
    }

    public boolean enabledByDefault() { return _defaultState; }

    public boolean enabledIn(int flags) { return (flags & getMask()) != 0; }

    public int getMask() { return (1 << ordinal()); }
};

/*
/**********************************************************
/* Minimal configuration state
/**********************************************************
 */

/**
```

```
     * Bit flag composed of bits that indicate which
     * {@link org.codehaus.jackson.JsonParser.Feature}s
     * are enabled.
     */
    protected int _features;

    /*
    /**********************************************************
    /* Minimal generic state
    /**********************************************************
     */

    /**
     * Last token retrieved via {@link #nextToken}, if any.
     * Null before the first call to <code>nextToken()</code>,
     * as well as if token has been explicitly cleared
     * (by call to {@link #clearCurrentToken})
     */
    protected JsonToken _currToken;

    /**
     * Last cleared token, if any: that is, value that was in
     * effect when {@link #clearCurrentToken} was called.
     */
    protected JsonToken _lastClearedToken;

    /*
    /**********************************************************
    /* Construction, init
    /**********************************************************
     */

    protected JsonParser() { }
    protected JsonParser(int features) {
        _features = features;
    }

    /**
     * Accessor for {@link ObjectCodec} associated with this
     * parser, if any. Codec is used by {@link #readValueAs(Class)}
     * method (and its variants).
     *
     * @since 1.3
     */
    public abstract ObjectCodec getCodec();

    /**
     * Setter that allows defining {@link ObjectCodec} associated with this
```

```
 * parser, if any. Codec is used by {@link #readValueAs(Class)}
 * method (and its variants).
 *
 * @since 1.3
 */
public abstract void setCodec(ObjectCodec c);

/**
 * @since 1.6
 */
public Version version() {
   return Version.unknownVersion();
}

/*
/**********************************************************
/* Closeable implementation
/**********************************************************
 */

/**
 * Closes the parser so that no further iteration or data access
 * can be made; will also close the underlying input source
 * if parser either <b>owns</b> the input source, or feature
 * {@link Feature#AUTO_CLOSE_SOURCE} is enabled.
 * Whether parser owns the input source depends on factory
 * method that was used to construct instance (so check
 * {@link org.codehaus.jackson.JsonFactory} for details,
 * but the general
 * idea is that if caller passes in closable resource (such
 * as {@link InputStream} or {@link Reader}) parser does NOT
 * own the source; but if it passes a reference (such as
 * {@link java.io.File} or {@link java.net.URL} and creates
 * stream or reader it does own them.
 */
public abstract void close() throws IOException;

/*
/**********************************************************
/* Buffer handling
/**********************************************************
 */

/**
 * Method that can be called to push back any content that
 * has been read but not consumed by the parser. This is usually
 * done after reading all content of interest using parser.
 * Content is released by writing it to given stream if possible;
```

```
 * if underlying input is byte-based it can released, if not (char-based)
 * it can not.
 *
 * @return -1 if the underlying content source is not byte based
 *    (that is, input can not be sent to {@link OutputStream};
 *    otherwise number of bytes released (0 if there was nothing to release)
 *
 * @throws IOException if write to stream threw exception
 *
 * @since 1.6
 */
public int releaseBuffered(OutputStream out) throws IOException
{
   return -1;
}

/**
 * Method that can be called to push back any content that
 * has been read but not consumed by the parser.
 * This is usually
 * done after reading all content of interest using parser.
 * Content is released by writing it to given writer if possible;
 * if underlying input is char-based it can released, if not (byte-based)
 * it can not.
 *
 * @return -1 if the underlying content source is not char-based
 *    (that is, input can not be sent to {@link Writer};
 *    otherwise number of chars released (0 if there was nothing to release)
 *
 * @throws IOException if write using Writer threw exception
 *
 * @since 1.6
 */
public int releaseBuffered(Writer w) throws IOException
{
   return -1;
}

/*
/**********************************************************
/* Public API, configuration
/**********************************************************
 */

/**
 * Method for enabling specified parser feature
 * (check {@link Feature} for list of features)
 *
```

```
 * @since 1.2
 */
public JsonParser enable(Feature f)
{
   _features |= f.getMask();
   return this;
}

/**
 * Method for disabling specified  feature
 * (check {@link Feature} for list of features)
 *
 * @since 1.2
 */
public JsonParser disable(Feature f)
{
   _features &= ~f.getMask();
   return this;
}

/**
 * Method for enabling or disabling specified feature
 * (check {@link Feature} for list of features)
 *
 * @since 1.2
 */
public JsonParser configure(Feature f, boolean state)
{
   if (state) {
      enableFeature(f);
   } else {
      disableFeature(f);
   }
   return this;
}

/**
 * Method for checking whether specified {@link Feature}
 * is enabled.
 *
 * @since 1.2
 */
public boolean isEnabled(Feature f) {
   return (_features & f.getMask()) != 0;
}

/** @deprecated Use {@link #configure} instead
 */
```

```
@SuppressWarnings("dep-ann")
public void setFeature(Feature f, boolean state) { configure(f, state); }

/** @deprecated Use {@link #enable(Feature)} instead
 */
@SuppressWarnings("dep-ann")
public void enableFeature(Feature f) { enable(f); }

/** @deprecated Use {@link #disable(Feature)} instead
 */
@SuppressWarnings("dep-ann")
public void disableFeature(Feature f) { disable(f); }

/** @deprecated Use {@link #isEnabled(Feature)} instead
 */
@SuppressWarnings("dep-ann")
public final boolean isFeatureEnabled(Feature f) { return isEnabled(f); }



/*
/**********************************************************
/* Public API, traversal
/**********************************************************
 */

/**
 * Main iteration method, which will advance stream enough
 * to determine type of the next token, if any. If none
 * remaining (stream has no content other than possible
 * white space before ending), null will be returned.
 *
 * @return Next token from the stream, if any found, or null
 *   to indicate end-of-input
 */
public abstract JsonToken nextToken()
   throws IOException, JsonParseException;

/**
 * Iteration method that will advance stream enough
 * to determine type of the next token that is a value type
 * (including JSON Array and Object start/end markers).
 * Or put another way, nextToken() will be called once,
 * and if {@link JsonToken#FIELD_NAME} is returned, another
 * time to get the value for the field.
 * Method is most useful for iterating over value entries
 * of JSON objects; field name will still be available
 * by calling {@link #getCurrentName} when parser points to
 * the value.
```

```
     *
     * @return Next non-field-name token from the stream, if any found,
     *   or null to indicate end-of-input (or, for non-blocking
     *   parsers, {@link JsonToken#NOT_AVAILABLE} if no tokens were
     *   available yet)
     *
     * @since 0.9.7
     */
    public JsonToken nextValue()
        throws IOException, JsonParseException
    {
        /* Implementation should be as trivial as follows; only
         * needs to change if we are to skip other tokens (for
         * example, if comments were exposed as tokens)
         */
        JsonToken t = nextToken();
        if (t == JsonToken.FIELD_NAME) {
            t = nextToken();
        }
        return t;
    }

    /**
     * Method that will skip all child tokens of an array or
     * object token that the parser currently points to,
     * iff stream points to
     * {@link JsonToken#START_OBJECT} or {@link JsonToken#START_ARRAY}.
     * If not, it will do nothing.
     * After skipping, stream will point to <b>matching</b>
     * {@link JsonToken#END_OBJECT} or {@link JsonToken#END_ARRAY}
     * (possibly skipping nested pairs of START/END OBJECT/ARRAY tokens
     * as well as value tokens).
     * The idea is that after calling this method, application
     * will call {@link #nextToken} to point to the next
     * available token, if any.
     */
    public abstract JsonParser skipChildren()
        throws IOException, JsonParseException;

    /**
     * Method that can be called to determine whether this parser
     * is closed or not. If it is closed, no new tokens can be
     * retrieved by calling {@link #nextToken} (and the underlying
     * stream may be closed). Closing may be due to an explicit
     * call to {@link #close} or because parser has encountered
     * end of input.
     */
    public abstract boolean isClosed();
```

```
/*
/**********************************************************
/* Public API, token accessors
/**********************************************************
 */

/**
 * Accessor to find which token parser currently points to, if any;
 * null will be returned if none.
 * If return value is non-null, data associated with the token
 * is available via other accessor methods.
 *
 * @return Type of the token this parser currently points to,
 *   if any: null before any tokens have been read, and
 *   after end-of-input has been encountered, as well as
 *   if the current token has been explicitly cleared.
 */
public JsonToken getCurrentToken() {
   return _currToken;
}

/**
 * Method for checking whether parser currently points to
 * a token (and data for that token is available).
 * Equivalent to check for <code>parser.getCurrentToken() != null</code>.
 *
 * @return True if the parser just returned a valid
 *   token via {@link #nextToken}; false otherwise (parser
 *   was just constructed, encountered end-of-input
 *   and returned null from {@link #nextToken}, or the token
 *   has been consumed)
 */
public boolean hasCurrentToken() {
   return _currToken != null;
}


/**
 * Method called to "consume" the current token by effectively
 * removing it so that {@link #hasCurrentToken} returns false, and
 * {@link #getCurrentToken} null).
 * Cleared token value can still be accessed by calling
 * {@link #getLastClearedToken} (if absolutely needed), but
 * usually isn't.
 *<p>
 * Method was added to be used by the optional data binder, since
 * it has to be able to consume last token used for binding (so that
```

```
     * it will not be used again).
     */
    public void clearCurrentToken() {
        if (_currToken != null) {
            _lastClearedToken = _currToken;
            _currToken = null;
        }
    }

    /**
     * Method that can be called to get the name associated with
     * the current token: for {@link JsonToken#FIELD_NAME}s it will
     * be the same as what {@link #getText} returns;
     * for field values it will be preceding field name;
     * and for others (array values, root-level values) null.
     */
    public abstract String getCurrentName()
        throws IOException, JsonParseException;

    /**
     * Method that can be used to access current parsing context reader
     * is in. There are 3 different types: root, array and object contexts,
     * with slightly different available information. Contexts are
     * hierarchically nested, and can be used for example for figuring
     * out part of the input document that correspond to specific
     * array or object (for highlighting purposes, or error reporting).
     * Contexts can also be used for simple xpath-like matching of
     * input, if so desired.
     */
    public abstract JsonStreamContext getParsingContext();

    /**
     * Method that return the <b>starting</b> location of the current
     * token; that is, position of the first character from input
     * that starts the current token.
     */
    public abstract JsonLocation getTokenLocation();

    /**
     * Method that returns location of the last processed character;
     * usually for error reporting purposes.
     */
    public abstract JsonLocation getCurrentLocation();

    /**
     * Method that can be called to get the last token that was
     * cleared using {@link #clearCurrentToken}. This is not necessarily
     * the latest token read.
```

```
     * Will return null if no tokens have been cleared,
     * or if parser has been closed.
     */
    public JsonToken getLastClearedToken() {
        return _lastClearedToken;
    }

    /**
     * Specialized accessor that can be used to verify that the current
     * token indicates start array (usually meaning that current token
     * is {@link JsonToken#START_ARRAY}) when start array is expected.
     * For some specialized parsers this can return true for other cases
     * as well; this is usually done to emulate arrays.
     *<p>
     * Default implementation is equivalent to:
     *<pre>
     *   getCurrentToken() == JsonToken.START_ARRAY
     *</pre>
     * but may be overridden by custom parser implementations.
     *
     * @return True if the current token can be considered as a
     *   start-array marker (such {@link JsonToken#START_ARRAY});
     *   false if not.
     *
     * @since 1.7
     */
    public boolean isExpectedStartArrayToken() {
        return getCurrentToken() == JsonToken.START_ARRAY;
    }

    /*
    /**********************************************************
    /* Public API, access to token information, text
    /**********************************************************
     */

    /**
     * Method for accessing textual representation of the current token;
     * if no current token (before first call to {@link #nextToken}, or
     * after encountering end-of-input), returns null.
     * Method can be called for any token type.
     */
    public abstract String getText()
        throws IOException, JsonParseException;

    /**
     * Method similar to {@link #getText}, but that will return
     * underlying (unmodifiable) character array that contains
```

```
 * textual value, instead of constructing a String object
 * to contain this information.
 * Note, however, that:
 *<ul>
 * <li>Textual contents are not guaranteed to start at
 *   index 0 (rather, call {@link #getTextOffset}) to
 *   know the actual offset
 *  </li>
 * <li>Length of textual contents may be less than the
 *  length of returned buffer: call {@link #getTextLength}
 *  for actual length of returned content.
 *  </li>
 * </ul>
 *<p>
 * Note that caller <b>MUST NOT</b> modify the returned
 * character array in any way -- doing so may corrupt
 * current parser state and render parser instance useless.
 *<p>
 * The only reason to call this method (over {@link #getText})
 * is to avoid construction of a String object (which
 * will make a copy of contents).
 */
public abstract char[] getTextCharacters()
   throws IOException, JsonParseException;

/**
 * Accessor used with {@link #getTextCharacters}, to know length
 * of String stored in returned buffer.
 *
 * @return Number of characters within buffer returned
 *   by {@link #getTextCharacters} that are part of
 *   textual content of the current token.
 */
public abstract int getTextLength()
   throws IOException, JsonParseException;

/**
 * Accessor used with {@link #getTextCharacters}, to know offset
 * of the first text content character within buffer.
 *
 * @return Offset of the first character within buffer returned
 *   by {@link #getTextCharacters} that is part of
 *   textual content of the current token.
 */
public abstract int getTextOffset()
   throws IOException, JsonParseException;

/**
```

```
     * Method that can be used to determine whether calling of
     * {@link #getTextCharacters} would be the most efficient
     * way to access textual content for the event parser currently
     * points to.
     *<p>
     * Default implementation simply returns false since only actual
     * implementation class has knowledge of its internal buffering
     * state.
     * Implementations are strongly encouraged to properly override
     * this method, to allow efficient copying of content by other
     * code.
     *
     * @return True if parser currently has character array that can
     *   be efficiently returned via {@link #getTextCharacters}; false
     *   means that it may or may not exist
     *
     * @since 1.6
     */
    public boolean hasTextCharacters() {
        return false;
    }

    /*
    /**********************************************************
    /* Public API, access to token information, numeric
    /**********************************************************
     */

    /**
     * Generic number value accessor method that will work for
     * all kinds of numeric values. It will return the optimal
     * (simplest/smallest possible) wrapper object that can
     * express the numeric value just parsed.
     */
    public abstract Number getNumberValue()
        throws IOException, JsonParseException;

    /**
     * If current token is of type
     * {@link JsonToken#VALUE_NUMBER_INT} or
     * {@link JsonToken#VALUE_NUMBER_FLOAT}, returns
     * one of {@link NumberType} constants; otherwise returns null.
     */
    public abstract NumberType getNumberType()
        throws IOException, JsonParseException;

    /**
     * Numeric accessor that can be called when the current
```

```
     * token is of type {@link JsonToken#VALUE_NUMBER_INT} and
     * it can be expressed as a value of Java byte primitive type.
     * It can also be called for {@link JsonToken#VALUE_NUMBER_FLOAT};
     * if so, it is equivalent to calling {@link #getDoubleValue}
     * and then casting; except for possible overflow/underflow
     * exception.
     *<p>
     * Note: if the resulting integer value falls outside range of
     * Java byte, a {@link JsonParseException}
     * will be thrown to indicate numeric overflow/underflow.
     */
    public byte getByteValue()
        throws IOException, JsonParseException
    {
        int value = getIntValue();
        // So far so good: but does it fit?
        if (value < MIN_BYTE_I || value > MAX_BYTE_I) {
            throw _constructError("Numeric value ("+getText()+") out of range of Java byte");
        }
        return (byte) value;
    }

    /**
     * Numeric accessor that can be called when the current
     * token is of type {@link JsonToken#VALUE_NUMBER_INT} and
     * it can be expressed as a value of Java short primitive type.
     * It can also be called for {@link JsonToken#VALUE_NUMBER_FLOAT};
     * if so, it is equivalent to calling {@link #getDoubleValue}
     * and then casting; except for possible overflow/underflow
     * exception.
     *<p>
     * Note: if the resulting integer value falls outside range of
     * Java short, a {@link JsonParseException}
     * will be thrown to indicate numeric overflow/underflow.
     */
    public short getShortValue()
        throws IOException, JsonParseException
    {
        int value = getIntValue();
        if (value < MIN_SHORT_I || value > MAX_SHORT_I) {
            throw _constructError("Numeric value ("+getText()+") out of range of Java short");
        }
        return (short) value;
    }

    /**
     * Numeric accessor that can be called when the current
     * token is of type {@link JsonToken#VALUE_NUMBER_INT} and
```

```
     * it can be expressed as a value of Java int primitive type.
     * It can also be called for {@link JsonToken#VALUE_NUMBER_FLOAT};
     * if so, it is equivalent to calling {@link #getDoubleValue}
     * and then casting; except for possible overflow/underflow
     * exception.
     *<p>
     * Note: if the resulting integer value falls outside range of
     * Java int, a {@link JsonParseException}
     * may be thrown to indicate numeric overflow/underflow.
     */
    public abstract int getIntValue()
        throws IOException, JsonParseException;

    /**
     * Numeric accessor that can be called when the current
     * token is of type {@link JsonToken#VALUE_NUMBER_INT} and
     * it can be expressed as a Java long primitive type.
     * It can also be called for {@link JsonToken#VALUE_NUMBER_FLOAT};
     * if so, it is equivalent to calling {@link #getDoubleValue}
     * and then casting to int; except for possible overflow/underflow
     * exception.
     *<p>
     * Note: if the token is an integer, but its value falls
     * outside of range of Java long, a {@link JsonParseException}
     * may be thrown to indicate numeric overflow/underflow.
     */
    public abstract long getLongValue()
        throws IOException, JsonParseException;

    /**
     * Numeric accessor that can be called when the current
     * token is of type {@link JsonToken#VALUE_NUMBER_INT} and
     * it can not be used as a Java long primitive type due to its
     * magnitude.
     * It can also be called for {@link JsonToken#VALUE_NUMBER_FLOAT};
     * if so, it is equivalent to calling {@link #getDecimalValue}
     * and then constructing a {@link BigInteger} from that value.
     */
    public abstract BigInteger getBigIntegerValue()
        throws IOException, JsonParseException;

    /**
     * Numeric accessor that can be called when the current
     * token is of type {@link JsonToken#VALUE_NUMBER_FLOAT} and
     * it can be expressed as a Java float primitive type.
     * It can also be called for {@link JsonToken#VALUE_NUMBER_INT};
     * if so, it is equivalent to calling {@link #getLongValue}
     * and then casting; except for possible overflow/underflow
```

```
     * exception.
     *<p>
     * Note: if the value falls
     * outside of range of Java float, a {@link JsonParseException}
     * will be thrown to indicate numeric overflow/underflow.
     */
    public abstract float getFloatValue()
        throws IOException, JsonParseException;

    /**
     * Numeric accessor that can be called when the current
     * token is of type {@link JsonToken#VALUE_NUMBER_FLOAT} and
     * it can be expressed as a Java double primitive type.
     * It can also be called for {@link JsonToken#VALUE_NUMBER_INT};
     * if so, it is equivalent to calling {@link #getLongValue}
     * and then casting; except for possible overflow/underflow
     * exception.
     *<p>
     * Note: if the value falls
     * outside of range of Java double, a {@link JsonParseException}
     * will be thrown to indicate numeric overflow/underflow.
     */
    public abstract double getDoubleValue()
        throws IOException, JsonParseException;

    /**
     * Numeric accessor that can be called when the current
     * token is of type {@link JsonToken#VALUE_NUMBER_FLOAT} or
     * {@link JsonToken#VALUE_NUMBER_INT}. No under/overflow exceptions
     * are ever thrown.
     */
    public abstract BigDecimal getDecimalValue()
        throws IOException, JsonParseException;

    /*
    /**********************************************************
    /* Public API, access to token information, other
    /**********************************************************
     */

    /**
     * Convenience accessor that can be called when the current
     * token is {@link JsonToken#VALUE_TRUE} or
     * {@link JsonToken#VALUE_FALSE}.
     *<p>
     * Note: if the token is not of above-mentioned boolean types,
  an integer, but its value falls
     * outside of range of Java long, a {@link JsonParseException}
```

```
     * may be thrown to indicate numeric overflow/underflow.
     *
     * @since 1.3
     */
    public boolean getBooleanValue()
        throws IOException, JsonParseException
    {
        if (_currToken == JsonToken.VALUE_TRUE) return true;
        if (_currToken == JsonToken.VALUE_FALSE) return false;
        throw new JsonParseException("Current token ("+_currToken+") not of boolean type", getCurrentLocation());
    }

    /**
     * Accessor that can be called if (and only if) the current token
     * is {@link JsonToken#VALUE_EMBEDDED_OBJECT}. For other token types,
     * null is returned.
     *<p>
     * Note: only some specialized parser implementations support
     * embedding of objects (usually ones that are facades on top
     * of non-streaming sources, such as object trees).
     *
     * @since 1.3
     */
    public Object getEmbeddedObject()
        throws IOException, JsonParseException
    {
        // By default we will always return null
        return null;
    }

    /*
    /**********************************************************
    /* Public API, access to token information, binary
    /**********************************************************
     */

    /**
     * Method that can be used to read (and consume -- results
     * may not be accessible using other methods after the call)
     * base64-encoded binary data
     * included in the current textual json value.
     * It works similar to getting String value via {@link #getText}
     * and decoding result (except for decoding part),
     * but should be significantly more performant.
     *<p>
     * Note that non-decoded textual contents of the current token
     * are not guaranteed to be accessible after this method
     * is called. Current implementation, for example, clears up
```

```
     * textual content during decoding.
     * Decoded binary content, however, will be retained until
     * parser is advanced to the next event.
     *
     * @param b64variant Expected variant of base64 encoded
     *   content (see {@link Base64Variants} for definitions
     *   of "standard" variants).
     *
     * @return Decoded binary data
     */
    public abstract byte[] getBinaryValue(Base64Variant b64variant) throws IOException, JsonParseException;

    /**
     * Convenience alternative to {@link #getBinaryValue(Base64Variant)}
     * that defaults to using
     * {@link Base64Variants#getDefaultVariant} as the default encoding.
     */
    public byte[] getBinaryValue() throws IOException, JsonParseException
    {
       return getBinaryValue(Base64Variants.getDefaultVariant());
    }

    /*
    /**********************************************************
    /* Public API, access to token information, coercion/conversion
    /**********************************************************
     */

    /**
     * Method that will try to convert value of current token to a
     * <b>int</b>.
     * Numbers are coerced using default Java rules; booleans convert to 0 (false)
     * and 1 (true), and Strings are parsed using default Java language integer
     * parsing rules.
     *<p>
     * If representation can not be converted to an int (including structured type
     * markers like start/end Object/Array)
     * default value of <b>0</b> will be returned; no exceptions are thrown.
     *
     * @since 1.6
     */
    public int getValueAsInt() throws IOException, JsonParseException {
       return getValueAsInt(0);
    }

    /**
     * Method that will try to convert value of current token to a
     * <b>int</b>.
```

```
     * Numbers are coerced using default Java rules; booleans convert to 0 (false)
     * and 1 (true), and Strings are parsed using default Java language integer
     * parsing rules.
     *<p>
     * If representation can not be converted to an int (including structured type
     * markers like start/end Object/Array)
     * specified <b>defaultValue</b> will be returned; no exceptions are thrown.
     *
     * @since 1.6
     */
    public int getValueAsInt(int defaultValue) throws IOException, JsonParseException {
        return defaultValue;
    }

    /**
     * Method that will try to convert value of current token to a
     * <b>long</b>.
     * Numbers are coerced using default Java rules; booleans convert to 0 (false)
     * and 1 (true), and Strings are parsed using default Java language integer
     * parsing rules.
     *<p>
     * If representation can not be converted to an int (including structured type
     * markers like start/end Object/Array)
     * default value of <b>0</b> will be returned; no exceptions are thrown.
     *
     * @since 1.6
     */
    public long getValueAsLong() throws IOException, JsonParseException {
        return getValueAsInt(0);
    }

    /**
     * Method that will try to convert value of current token to a
     * <b>long</b>.
     * Numbers are coerced using default Java rules; booleans convert to 0 (false)
     * and 1 (true), and Strings are parsed using default Java language integer
     * parsing rules.
     *<p>
     * If representation can not be converted to an int (including structured type
     * markers like start/end Object/Array)
     * specified <b>defaultValue</b> will be returned; no exceptions are thrown.
     *
     * @since 1.6
     */
    public long getValueAsLong(long defaultValue) throws IOException, JsonParseException {
        return defaultValue;
    }
```

```
/**
 * Method that will try to convert value of current token to a Java
 * <b>double</b>.
 * Numbers are coerced using default Java rules; booleans convert to 0.0 (false)
 * and 1.0 (true), and Strings are parsed using default Java language integer
 * parsing rules.
 *<p>
 * If representation can not be converted to an int (including structured types
 * like Objects and Arrays),
 * default value of <b>0.0</b> will be returned; no exceptions are thrown.
 *
 * @since 1.6
 */
public double getValueAsDouble() throws IOException, JsonParseException {
   return getValueAsDouble(0.0);
}

/**
 * Method that will try to convert value of current token to a
 * Java <b>double</b>.
 * Numbers are coerced using default Java rules; booleans convert to 0.0 (false)
 * and 1.0 (true), and Strings are parsed using default Java language integer
 * parsing rules.
 *<p>
 * If representation can not be converted to an int (including structured types
 * like Objects and Arrays),
 * specified <b>defaultValue</b> will be returned; no exceptions are thrown.
 *
 * @since 1.6
 */
public double getValueAsDouble(double defaultValue) throws IOException, JsonParseException {
   return defaultValue;
}

/**
 * Method that will try to convert value of current token to a
 * <b>boolean</b>.
 * JSON booleans map naturally; integer numbers other than 0 map to true, and
 * 0 maps to false
 * and Strings 'true' and 'false' map to corresponding values.
 *<p>
 * If representation can not be converted to a boolean value (including structured types
 * like Objects and Arrays),
 * default value of <b>false</b> will be returned; no exceptions are thrown.
 *
 * @since 1.7
 */
public boolean getValueAsBoolean() throws IOException, JsonParseException {
```

```
    return getValueAsBoolean(false);
}

/**
 * Method that will try to convert value of current token to a
 * <b>boolean</b>.
 * JSON booleans map naturally; integer numbers other than 0 map to true, and
 * 0 maps to false
 * and Strings 'true' and 'false' map to corresponding values.
 *<p>
 * If representation can not be converted to a boolean value (including structured types
 * like Objects and Arrays),
 * specified <b>defaultValue</b> will be returned; no exceptions are thrown.
 *
 * @since 1.7
 */
public boolean getValueAsBoolean(boolean defaultValue) throws IOException, JsonParseException {
    return defaultValue;
}

/*
/**********************************************************
/* Public API, optional data binding functionality
/**********************************************************
 */

/**
 * Method to deserialize JSON content into a non-container
 * type (it can be an array type, however): typically a bean, array
 * or a wrapper type (like {@link java.lang.Boolean}).
 * <b>Note</b>: method can only be called if the parser has
 * an object codec assigned; this is true for parsers constructed
 * by {@link org.codehaus.jackson.map.MappingJsonFactory} but
 * not for {@link JsonFactory} (unless its <code>setCodec</code>
 * method has been explicitly called).
 *<p>
 * This method may advance the event stream, for structured types
 * the current token will be the closing end marker (END_ARRAY,
 * END_OBJECT) of the bound structure. For non-structured Json types
 * (and for {@link JsonToken#VALUE_EMBEDDED_OBJECT})
 * stream is not advanced.
 *<p>
 * Note: this method should NOT be used if the result type is a
 * container ({@link java.util.Collection} or {@link java.util.Map}.
 * The reason is that due to type erasure, key and value types
 * can not be introspected when using this method.
 */
public <T> T readValueAs(Class<T> valueType)
```

```
    throws IOException, JsonProcessingException
  {
    ObjectCodec codec = getCodec();
    if (codec == null) {
      throw new IllegalStateException("No ObjectCodec defined for the parser, can not deserialize JSON into Java
objects");
    }
    return codec.readValue(this, valueType);
  }

  /**
   * Method to deserialize JSON content into a Java type, reference
   * to which is passed as argument. Type is passed using so-called
   * "super type token"
   * and specifically needs to be used if the root type is a
   * parameterized (generic) container type.
   * <b>Note</b>: method can only be called if the parser has
   * an object codec assigned; this is true for parsers constructed
   * by {@link org.codehaus.jackson.map.MappingJsonFactory} but
   * not for {@link JsonFactory} (unless its <code>setCodec</code>
   * method has been explicitly called).
   *<p>
   * This method may advance the event stream, for structured types
   * the current token will be the closing end marker (END_ARRAY,
   * END_OBJECT) of the bound structure. For non-structured Json types
   * (and for {@link JsonToken#VALUE_EMBEDDED_OBJECT})
   * stream is not advanced.
   */
  @SuppressWarnings("unchecked")
  public <T> T readValueAs(TypeReference<?> valueTypeRef)
    throws IOException, JsonProcessingException
  {
    ObjectCodec codec = getCodec();
    if (codec == null) {
      throw new IllegalStateException("No ObjectCodec defined for the parser, can not deserialize JSON into Java
objects");
    }
    /* Ugh. Stupid Java type erasure... can't just chain call,s
     * must cast here also.
     */
    return (T) codec.readValue(this, valueTypeRef);
  }

  /**
   * Method to deserialize JSON content into equivalent "tree model",
   * represented by root {@link JsonNode} of resulting model.
   * For JSON Arrays it will an array node (with child nodes),
   * for objects object node (with child nodes), and for other types
```

```
     * matching leaf node type
     */
    public JsonNode readValueAsTree()
        throws IOException, JsonProcessingException
    {
        ObjectCodec codec = getCodec();
        if (codec == null) {
            throw new IllegalStateException("No ObjectCodec defined for the parser, can not deserialize JSON into
JsonNode tree");
        }
        return codec.readTree(this);
    }

    /*
    /**********************************************************
    /* Internal methods
    /**********************************************************
     */

    /**
     * Helper method for constructing {@link JsonParseException}s
     * based on current state of the parser
     */
    protected JsonParseException _constructError(String msg)
    {
        return new JsonParseException(msg, getCurrentLocation());
    }
}
package org.codehaus.jackson;

/**
* Intermediate base class for all problems encountered when
* processing (parsing, generating) JSON content
* that are not pure I/O problems.
* Regular {@link java.io.IOException}s will be passed through as is.
* Sub-class of {@link java.io.IOException} for convenience.
*/
public class JsonProcessingException
    extends java.io.IOException
{
    final static long serialVersionUID = 123; // Stupid eclipse...

    protected JsonLocation mLocation;

    protected JsonProcessingException(String msg, JsonLocation loc, Throwable rootCause)
    {
        /* Argh. IOException(Throwable,String) is only available starting
         * with JDK 1.6...
```

```
     */
    super(msg);
    if (rootCause != null) {
        initCause(rootCause);
    }
    mLocation = loc;
}

protected JsonProcessingException(String msg)
{
    super(msg);
}

protected JsonProcessingException(String msg, JsonLocation loc)
{
    this(msg, loc, null);
}

protected JsonProcessingException(String msg, Throwable rootCause)
{
    this(msg, null, rootCause);
}

protected JsonProcessingException(Throwable rootCause)
{
    this(null, null, rootCause);
}

public JsonLocation getLocation()
{
    return mLocation;
}

/**
 * Default method overridden so that we can add location information
 */
@Override
public String getMessage()
{
    String msg = super.getMessage();
    if (msg == null) {
        msg = "N/A";
    }
    JsonLocation loc = getLocation();
    if (loc != null) {
        StringBuilder sb = new StringBuilder();
        sb.append(msg);
        sb.append('\n');
```

```
            sb.append(" at ");
            sb.append(loc.toString());
            return sb.toString();
        }
        return msg;
    }

    @Override
    public String toString() {
        return getClass().getName()+": "+getMessage();
    }
}
package org.codehaus.jackson;

import org.codehaus.jackson.annotate.JsonCreator;
import org.codehaus.jackson.annotate.JsonProperty;

/**
* Object that encapsulates Location information used for reporting
* parsing (or potentially generation) errors, as well as current location
* within input streams.
*/
public class JsonLocation
    implements java.io.Serializable // as per [JACKSON-168]
{
    private static final long serialVersionUID = 1L;

    /**
     * Shared immutable "N/A location" that can be returned to indicate
     * that no location information is available
     *
     * @since 1.3
     */
    public final static JsonLocation NA = new JsonLocation("N/A", -1L, -1L, -1, -1);

    final long _totalBytes;
    final long _totalChars;

    final int _lineNr;
    final int _columnNr;

    /**
     * Displayable description for input source: file path, url
     */
    final Object _sourceRef;

    public JsonLocation(Object srcRef, long totalChars, int lineNr, int colNr)
    {
```

```
    /* Unfortunately, none of legal encodings are straight single-byte
     * encodings. Could determine offset for UTF-16/UTF-32, but the
     * most important one is UTF-8...
     * so for now, we'll just not report any real byte count
     */
    this(srcRef, -1L, totalChars, lineNr, colNr);
}

@JsonCreator
public JsonLocation(@JsonProperty("sourceRef") Object sourceRef,
                    @JsonProperty("byteOffset") long totalBytes,
                    @JsonProperty("charOffset") long totalChars,
                    @JsonProperty("lineNr") int lineNr,
                    @JsonProperty("columnNr") int columnNr)
{
    _sourceRef = sourceRef;
    _totalBytes = totalBytes;
    _totalChars = totalChars;
    _lineNr = lineNr;
    _columnNr = columnNr;
}

/**
 * Reference to the original resource being read, if one available.
 * For example, when a parser has been constructed by passing
 * a {@link java.io.File} instance, this method would return
 * that File. Will return null if no such reference is available,
 * for example when {@link java.io.InputStream} was used to
 * construct the parser instance.
 */
public Object getSourceRef() { return _sourceRef; }

/**
 * @return Line number of the location (1-based)
 */
public int getLineNr() { return _lineNr; }

/**
 * @return Column number of the location (1-based)
 */
public int getColumnNr() { return _columnNr; }

/**
 * @return Character offset within underlying stream, reader or writer,
 *   if available; -1 if not.
 */
public long getCharOffset() { return _totalChars; }
```

```
/**
 * @return Byte offset within underlying stream, reader or writer,
 *   if available; -1 if not.
 */
public long getByteOffset()
{
   return _totalBytes;
}

@Override
public String toString()
{
   StringBuilder sb = new StringBuilder(80);
   sb.append("[Source: ");
   if (_sourceRef == null) {
      sb.append("UNKNOWN");
   } else {
      sb.append(_sourceRef.toString());
   }
   sb.append("; line: ");
   sb.append(_lineNr);
   sb.append(", column: ");
   sb.append(_columnNr);
   sb.append(']');
   return sb.toString();
}

@Override
public int hashCode()
{
   int hash = (_sourceRef == null) ? 1 : _sourceRef.hashCode();
   hash ^= _lineNr;
   hash += _columnNr;
   hash ^= (int) _totalChars;
   hash += (int) _totalBytes;
   return hash;
}

@Override
public boolean equals(Object other)
{
   if (other == this) return true;
   if (other == null) return false;
   if (!(other instanceof JsonLocation)) return false;
   JsonLocation otherLoc = (JsonLocation) other;

   if (_sourceRef == null) {
      if (otherLoc._sourceRef != null) return false;
```

```
        } else if (!_sourceRef.equals(otherLoc._sourceRef)) return false;

        return (_lineNr == otherLoc._lineNr)
            && (_columnNr == otherLoc._columnNr)
            && (_totalChars == otherLoc._totalChars)
            && (getByteOffset() == otherLoc.getByteOffset())
            ;
    }
}
package org.codehaus.jackson;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.*;

/**
* Base class for all JSON nodes, which form the basis of JSON
* Tree Model that Jackson implements.
* One way to think of these nodes is to consider them
* similar to DOM nodes in XML DOM trees.
*<p>
* As a general design rule, most accessors ("getters") are included
* in this base class, to allow for traversing structure without
* type casts. Most mutators, however, need to be accessed through
* specific sub-classes. This seems sensible because proper type
* information is generally available when building or modifying
* trees, but less often when reading a tree (newly built from
* parsed JSON content).
*<p>
* Actual concrete sub-classes can be found from package
* {@link org.codehaus.jackson.node}, which is in 'mapper' jar
* (whereas this class is in 'core' jar, since it is declared as
* nominal type for operations in {@link ObjectCodec}
*/
public abstract class JsonNode
    implements Iterable<JsonNode>
{
    protected final static List<JsonNode> NO_NODES = Collections.emptyList();
    protected final static List<String> NO_STRINGS = Collections.emptyList();

    protected JsonNode() { }

    /*
    /**********************************************************
    /* Public API, type introspection
    /**********************************************************
     */
```

```
// // First high-level division between values, containers and "missing"

/**
 * Method that returns true for all value nodes: ones that
 * are not containers, and that do not represent "missing" nodes
 * in the path. Such value nodes represent String, Number, Boolean
 * and null values from JSON.
 *<p>
 * Note: one and only one of methods {@link #isValueNode},
 * {@link #isContainerNode} and {@link #isMissingNode} ever
 * returns true for any given node.
 */
public boolean isValueNode() { return false; }

/**
 * Method that returns true for container nodes: Arrays and Objects.
 *<p>
 * Note: one and only one of methods {@link #isValueNode},
 * {@link #isContainerNode} and {@link #isMissingNode} ever
 * returns true for any given node.
 */
public boolean isContainerNode() { return false; }

/**
 * Method that returns true for "virtual" nodes which represent
 * missing entries constructed by path accessor methods when
 * there is no actual node matching given criteria.
 *<p>
 * Note: one and only one of methods {@link #isValueNode},
 * {@link #isContainerNode} and {@link #isMissingNode} ever
 * returns true for any given node.
 */
public boolean isMissingNode() { return false; }

// // Then more specific type introspection
// // (along with defaults to be overridden)

/**
 * @return True if this node represents Json Array
 */
public boolean isArray() { return false; }

/**
 * @return True if this node represents Json Object
 */
public boolean isObject() { return false; }
```

```
/**
 * Method that can be used to check if the node is a wrapper
 * for a POJO ("Plain Old Java Object" aka "bean".
 * Returns true only for
 * instances of {@link org.codehaus.jackson.node.POJONode}.
 *
 * @return True if this node wraps a POJO
 */
public boolean isPojo() { return false; }

/**
 * @return True if this node represents a numeric Json
 *   value
 */
public boolean isNumber() { return false; }

/**
 * @return True if this node represents an integral (integer)
 *   numeric Json value
 */
public boolean isIntegralNumber() { return false; }

/**
 * @return True if this node represents a non-integral
 *   numeric Json value
 */
public boolean isFloatingPointNumber() { return false; }

/**
 * @return True if this node represents an integral
 *   numeric Json value that withs in Java int value space
 */
public boolean isInt() { return false; }

/**
 * @return True if this node represents an integral
 *   numeric Json value that fits in Java long value space
 *   (but not int value space, i.e. {@link #isInt} returns false)
 */
public boolean isLong() { return false; }

public boolean isDouble() { return false; }
public boolean isBigDecimal() { return false; }
public boolean isBigInteger() { return false; }

public boolean isTextual() { return false; }

/**
```

```
     * Method that can be used to check if this node was created from
     * Json boolean value (literals "true" and "false").
     */
    public boolean isBoolean() { return false; }

    /**
     * Method that can be used to check if this node was created from
     * Json liternal null value.
     */
    public boolean isNull() { return false; }

    /**
     * Method that can be used to check if this node represents
     * binary data (Base64 encoded). Although this will be externally
     * written as Json String value, {@link #isTextual} will
     * return false if this method returns true.
     *
     * @return True if this node represents base64 encoded binary data
     */
    public boolean isBinary() { return false; }

    /**
     * Method that can be used for efficient type detection
     * when using stream abstraction for traversing nodes.
     * Will return the first {@link JsonToken} that equivalent
     * stream event would produce (for most nodes there is just
     * one token but for structured/container types multiple)
     *
     * @since 1.3
     */
    public abstract JsonToken asToken();

    /**
     * If this node is a numeric type (as per {@link #isNumber}),
     * returns native type that node uses to store the numeric
     * value.
     */
    public abstract JsonParser.NumberType getNumberType();

    /*
    /**********************************************************
    /* Public API, straight value access
    /**********************************************************
     */

    /**
     * Method to use for accessing String values.
     * Does <b>NOT</b> do any conversions for non-String value nodes;
```

```
 * for non-String values (ones for which {@link #isTextual} returns
 * false) null will be returned.
 * For String values, null is never returned (but empty Strings may be)
 *
 * @return Textual value this node contains, iff it is a textual
 *   json node (comes from Json String value entry)
 */
public String getTextValue() { return null; }

/**
 * Method to use for accessing binary content of binary nodes (nodes
 * for which {@link #isBinary} returns true); or for Text Nodes
 * (ones for which {@link #getTextValue} returns non-null value),
 * to read decoded base64 data.
 * For other types of nodes, returns null.
 *
 * @return Binary data this node contains, iff it is a binary
 *   node; null otherwise
 */
public byte[] getBinaryValue() throws IOException
{
   return null;
}

/**
 * Method to use for accessing JSON boolean values (value
 * literals 'true' and 'false').
 * For other types, always returns false.
 *
 * @return Textual value this node contains, iff it is a textual
 *   json node (comes from Json String value entry)
 */
public boolean getBooleanValue() { return false; }

/**
 * Returns numeric value for this node, <b>if and only if</b>
 * this node is numeric ({@link #isNumber} returns true); otherwise
 * returns null
 *
 * @return Number value this node contains, if any (null for non-number
 *   nodes).
 */
public Number getNumberValue() { return null; }

/**
 * Returns integer value for this node, <b>if and only if</b>
 * this node is numeric ({@link #isNumber} returns true). For other
 * types returns 0.
```

```
     * For floating-point numbers, value is truncated using default
     * Java coercion, similar to how cast from double to int operates.
     *
     * @return Integer value this node contains, if any; 0 for non-number
     *   nodes.
     */
    public int getIntValue() { return 0; }

    public long getLongValue() { return 0L; }
    public double getDoubleValue() { return 0.0; }
    public BigDecimal getDecimalValue() { return BigDecimal.ZERO; }
    public BigInteger getBigIntegerValue() { return BigInteger.ZERO; }

    /**
     * Method for accessing value of the specified element of
     * an array node. For other nodes, null is always returned.
     *<p>
     * For array nodes, index specifies
     * exact location within array and allows for efficient iteration
     * over child elements (underlying storage is guaranteed to
     * be efficiently indexable, i.e. has random-access to elements).
     * If index is less than 0, or equal-or-greater than
     * <code>node.size()</code>, null is returned; no exception is
     * thrown for any index.
     *
     * @return Node that represent value of the specified element,
     *   if this node is an array and has specified element.
     *   Null otherwise.
     */
    public JsonNode get(int index) { return null; }

    /**
     * Method for accessing value of the specified field of
     * an object node. If this node is not an object (or it
     * does not have a value for specified field name), or
     * if there is no field with such name, null is returned.
     *
     * @return Node that represent value of the specified field,
     *   if this node is an object and has value for the specified
     *   field. Null otherwise.
     */
    public JsonNode get(String fieldName) { return null; }

    /*
    /**********************************************************
    /* Public API, value access with conversion(s)/coercion(s)
    /**********************************************************
     */
```

```
/**
 * Method that will return valid String representation of
 * the container value, if the node is a value node
 * (method {@link #isValueNode} returns true), otherwise null.
 *<p>
 * Note: to serialize nodes of any type, you should call
 * {@link #toString} instead.
 */
public abstract String getValueAsText();

/**
 * Method that will try to convert value of this node to a Java <b>int</b>.
 * Numbers are coerced using default Java rules; booleans convert to 0 (false)
 * and 1 (true), and Strings are parsed using default Java language integer
 * parsing rules.
 *<p>
 * If representation can not be converted to an int (including structured types
 * like Objects and Arrays),
 * default value of <b>0</b> will be returned; no exceptions are thrown.
 *
 * @since 1.6
 */
public int getValueAsInt() {
   return getValueAsInt(0);
}

/**
 * Method that will try to convert value of this node to a Java <b>int</b>.
 * Numbers are coerced using default Java rules; booleans convert to 0 (false)
 * and 1 (true), and Strings are parsed using default Java language integer
 * parsing rules.
 *<p>
 * If representation can not be converted to an int (including structured types
 * like Objects and Arrays),
 * specified <b>defaultValue</b> will be returned; no exceptions are thrown.
 *
 * @since 1.6
 */
public int getValueAsInt(int defaultValue) {
   return defaultValue;
}

/**
 * Method that will try to convert value of this node to a Java <b>long</b>.
 * Numbers are coerced using default Java rules; booleans convert to 0 (false)
 * and 1 (true), and Strings are parsed using default Java language integer
 * parsing rules.
```

```
 *<p>
 * If representation can not be converted to an long (including structured types
 * like Objects and Arrays),
 * default value of <b>0</b> will be returned; no exceptions are thrown.
 *
 * @since 1.6
 */
public long getValueAsLong() {
   return getValueAsInt(0);
}

/**
 * Method that will try to convert value of this node to a Java <b>long</b>.
 * Numbers are coerced using default Java rules; booleans convert to 0 (false)
 * and 1 (true), and Strings are parsed using default Java language integer
 * parsing rules.
 *<p>
 * If representation can not be converted to an long (including structured types
 * like Objects and Arrays),
 * specified <b>defaultValue</b> will be returned; no exceptions are thrown.
 *
 * @since 1.6
 */
public long getValueAsLong(long defaultValue) {
   return defaultValue;
}

/**
 * Method that will try to convert value of this node to a Java <b>double</b>.
 * Numbers are coerced using default Java rules; booleans convert to 0.0 (false)
 * and 1.0 (true), and Strings are parsed using default Java language integer
 * parsing rules.
 *<p>
 * If representation can not be converted to an int (including structured types
 * like Objects and Arrays),
 * default value of <b>0.0</b> will be returned; no exceptions are thrown.
 *
 * @since 1.6
 */
public double getValueAsDouble() {
   return getValueAsDouble(0.0);
}

/**
 * Method that will try to convert value of this node to a Java <b>double</b>.
 * Numbers are coerced using default Java rules; booleans convert to 0.0 (false)
 * and 1.0 (true), and Strings are parsed using default Java language integer
 * parsing rules.
```

```
 *<p>
 * If representation can not be converted to an int (including structured types
 * like Objects and Arrays),
 * specified <b>defaultValue</b> will be returned; no exceptions are thrown.
 *
 * @since 1.6
 */
public double getValueAsDouble(double defaultValue) {
   return defaultValue;
}

/**
 * Method that will try to convert value of this node to a Java <b>boolean</b>.
 * JSON booleans map naturally; integer numbers other than 0 map to true, and
 * 0 maps to false
 * and Strings 'true' and 'false' map to corresponding values.
 *<p>
 * If representation can not be converted to a boolean value (including structured types
 * like Objects and Arrays),
 * default value of <b>false</b> will be returned; no exceptions are thrown.
 *
 * @since 1.7
 */
public boolean getValueAsBoolean() {
   return getValueAsBoolean(false);
}

/**
 * Method that will try to convert value of this node to a Java <b>boolean</b>.
 * JSON booleans map naturally; integer numbers other than 0 map to true, and
 * 0 maps to false
 * and Strings 'true' and 'false' map to corresponding values.
 *<p>
 * If representation can not be converted to a boolean value (including structured types
 * like Objects and Arrays),
 * specified <b>defaultValue</b> will be returned; no exceptions are thrown.
 *
 * @since 1.7
 */
public boolean getValueAsBoolean(boolean defaultValue) {
   return defaultValue;
}

/*
/**********************************************************
/* Public API, value find / existence check methods
/**********************************************************
 */
```

```
/**
 * Method that allows checking whether this node is JSON Object node
 * and contains value for specified property. If this is the case
 * (including properties with explicit null values), returns true;
 * otherwise returns false.
 *<p>
 * This method is equivalent to:
 *<pre>
 *   node.get(fieldName) != null
 *</pre>
 * (since return value of get() is node, not value node contains)
 *
 * @param fieldName Name of element to check
 *
 * @return True if this node is a JSON Object node, and has a property
 *   entry with specified name (with any value, including null value)
 *
 * @since 1.6
 */
public boolean has(String fieldName) {
   return get(fieldName) != null;
}

/**
 * Method that allows checking whether this node is JSON Array node
 * and contains a value for specified index
 * If this is the case
 * (including case of specified indexing having null as value), returns true;
 * otherwise returns false.
 *<p>
 * Note: array element indexes are 0-based.
 *<p>
 * This method is equivalent to:
 *<pre>
 *   node.get(index) != null
 *</pre>
 *
 * @param index Index to check
 *
 * @return True if this node is a JSON Object node, and has a property
 *   entry with specified name (with any value, including null value)
 *
 * @since 1.6
 */
public boolean has(int index) {
   return get(index) != null;
}
```

```
/**
 * Method for finding a JSON Object field with specified name in this
 * node or its child nodes, and returning value it has.
 * If no matching field is found in this node or its descendants, returns null.
 *
 * @param fieldName Name of field to look for
 *
 * @return Value of first matching node found, if any; null if none
 *
 * @since 1.6
 */
public abstract JsonNode findValue(String fieldName);

/**
 * Method for finding JSON Object fields with specified name, and returning
 * found ones as a List. Note that sub-tree search ends if a field is found,
 * so possible children of result nodes are <b>not</b> included.
 * If no matching fields are found in this node or its descendants, returns
 * an empty List.
 *
 * @param fieldName Name of field to look for
 *
 * @since 1.6
 */
public final List<JsonNode> findValues(String fieldName)
{
   List<JsonNode> result = findValues(fieldName, null);
   if (result == null) {
      return Collections.emptyList();
   }
   return result;
}

/**
 * Similar to {@link #findValues}, but will additionally convert
 * values into Strings, calling {@link #getValueAsText}.
 *
 * @since 1.6
 */
public final List<String> findValuesAsText(String fieldName)
{
   List<String> result = findValuesAsText(fieldName, null);
   if (result == null) {
      return Collections.emptyList();
   }
   return result;
}
```

```
/**
 * Method similar to {@link #findValue}, but that will return a
 * "missing node" instead of null if no field is found. Missing node
 * is a specific kind of node for which {@link #isMissingNode}
 * returns true; and all value access methods return empty or
 * missing value.
 *
 * @param fieldName Name of field to look for
 *
 * @return Value of first matching node found; or if not found, a
 *    "missing node" (non-null instance that has no value)
 *
 * @since 1.6
 */
public abstract JsonNode findPath(String fieldName);

/**
 * Method for finding a JSON Object that contains specified field,
 * within this node or its descendants.
 * If no matching field is found in this node or its descendants, returns null.
 *
 * @param fieldName Name of field to look for
 *
 * @return Value of first matching node found, if any; null if none
 *
 * @since 1.6
 */
public abstract JsonNode findParent(String fieldName);

/**
 * Method for finding a JSON Object that contains specified field,
 * within this node or its descendants.
 * If no matching field is found in this node or its descendants, returns null.
 *
 * @param fieldName Name of field to look for
 *
 * @return Value of first matching node found, if any; null if none
 *
 * @since 1.6
 */
public final List<JsonNode> findParents(String fieldName)
{
   List<JsonNode> result = findParents(fieldName, null);
   if (result == null) {
      return Collections.emptyList();
   }
   return result;
```

```
}

public abstract List<JsonNode> findValues(String fieldName, List<JsonNode> foundSoFar);
public abstract List<String> findValuesAsText(String fieldName, List<String> foundSoFar);
public abstract List<JsonNode> findParents(String fieldName, List<JsonNode> foundSoFar);

/*
/**********************************************************
/* Public API, deprecated accessor
/**********************************************************
 */

/**
 * Alias for {@link #get(String)}.
 *
 * @deprecated Use {@link #get(String)} instead.
 */
@Deprecated
public final JsonNode getFieldValue(String fieldName) { return get(fieldName); }

/**
 * Alias for {@link #get(int)}.
 *
 * @deprecated Use {@link #get(int)} instead.
 */
@Deprecated
public final JsonNode getElementValue(int index) { return get(index); }

/*
/**********************************************************
/* Public API, container access
/**********************************************************
 */

/**
 * Method that returns number of child nodes this node contains:
 * for Array nodes, number of child elements, for Object nodes,
 * number of fields, and for all other nodes 0.
 *
 * @return For non-container nodes returns 0; for arrays number of
 *   contained elements, and for objects number of fields.
 */
public int size() { return 0; }

/**
 * Same as calling {@link #getElements}; implemented so that
 * convenience "for-each" loop can be used for looping over elements
 * of JSON Array constructs.
```

```
 */
public final Iterator<JsonNode> iterator() { return getElements(); }

/**
 * Method for accessing all value nodes of this Node, iff
 * this node is a JSON Array or Object node. In case of Object node,
 * field names (keys) are not included, only values.
 * For other types of nodes, returns empty iterator.
 */
public Iterator<JsonNode> getElements() { return NO_NODES.iterator(); }

/**
 * Method for accessing names of all fields for this Node, iff
 * this node is a JSON Object node.
 */
public Iterator<String> getFieldNames() { return NO_STRINGS.iterator(); }

/*
/**********************************************************
/* Public API, path handling
/**********************************************************
 */

/**
 * This method is similar to {@link #get(String)}, except
 * that instead of returning null if no such value exists (due
 * to this node not being an object, or object not having value
 * for the specified field),
 * a "missing node" (node that returns true for
 * {@link #isMissingNode}) will be returned. This allows for
 * convenient and safe chained access via path calls.
 */
public abstract JsonNode path(String fieldName);

/**
 * Alias of {@link #path(String)}.
 *
 * @deprecated Use {@link #path(String)} instead
 */
@Deprecated
public final JsonNode getPath(String fieldName) { return path(fieldName); }

/**
 * This method is similar to {@link #get(int)}, except
 * that instead of returning null if no such element exists (due
 * to index being out of range, or this node not being an array),
 * a "missing node" (node that returns true for
 * {@link #isMissingNode}) will be returned. This allows for
```

```
     * convenient and safe chained access via path calls.
     */
    public abstract JsonNode path(int index);

    /**
     * Alias of {@link #path(int)}.
     *
     * @deprecated Use {@link #path(int)} instead
     */
    @Deprecated
    public final JsonNode getPath(int index) { return path(index); }

    /*
    /**********************************************************
    /* Public API, serialization
    /**********************************************************
     */

    /**
     * Method that can be called to serialize this node and
     * all of its descendants using specified JSON generator.
     *
     * @deprecated Use methods that are part of {@link JsonGenerator}
     *   or {@link org.codehaus.jackson.map.ObjectMapper}
     *   instead.
     */
    @Deprecated
    public abstract void writeTo(JsonGenerator jg)
       throws IOException, JsonGenerationException;

    /*
    /**********************************************************
    /* Public API: converting to/from Streaming API
    /**********************************************************
     */


    /**
     * Method for constructing a {@link JsonParser} instance for
     * iterating over contents of the tree that this
     * node is root of.
     * Functionally equivalent to first serializing tree
     * using {@link #writeTo} and then re-parsing but much
     * more efficient.
     */
    public abstract JsonParser traverse();

    /*
```

```
  /**********************************************************
  /* Overridden standard methods
  /**********************************************************
   */

  /**
   *<p>
   * Note: marked as abstract to ensure all implementation
   * classes define it properly.
   */
  @Override
  public abstract String toString();

  /**
   * Equality for node objects is defined as full (deep) value
   * equality. This means that it is possible to compare complete
   * JSON trees for equality by comparing equality of root nodes.
   *<p>
   * Note: marked as abstract to ensure all implementation
   * classes define it properly and not rely on definition
   * from {@link java.lang.Object}.
   */
  @Override
  public abstract boolean equals(Object o);
}
package org.codehaus.jackson;

/**
* Interface that those Jackson components that are explicitly versioned will implement.
* Intention is to allow both plug-in components (custom extensions) and applications and
* frameworks that use Jackson to detect exact version of Jackson in use.
* This may be useful for example for ensuring that proper Jackson version is deployed
* (beyond mechanisms that deployment system may have), as well as for possible
* workarounds.
*
* @since 1.6
*/
public interface Versioned {
  /**
   * Method called to detect version of the component that implements this interface;
   * returned version should never be null, but may return specific "not available"
   * instance (see {@link Version} for details).
   */
  public Version version();
}
package org.codehaus.jackson;

/**
```

```
* Object that encapsulates version information of a component,
* and is return by {@link Versioned#version}.
*
* @since 1.6
*/
public class Version
  implements Comparable<Version>
{
  private final static Version UNKNOWN_VERSION = new Version(0, 0, 0, null);

  protected final int _majorVersion;

  protected final int _minorVersion;

  protected final int _patchLevel;

  /**
   * Additional information for snapshot versions; null for non-snapshot
   * (release) versions.
   */
  protected final String _snapshotInfo;

  public Version(int major, int minor, int patchLevel,
        String snapshotInfo)
  {
    _majorVersion = major;
    _minorVersion = minor;
    _patchLevel = patchLevel;
    _snapshotInfo = snapshotInfo;
  }

  /**
   * Method returns canonical "not known" version, which is used as version
   * in cases where actual version information is not known (instead of null).
   */
  public static Version unknownVersion() { return UNKNOWN_VERSION; }

  public boolean isUknownVersion() { return (this == UNKNOWN_VERSION); }
  public boolean isSnapshot() { return (_snapshotInfo != null && _snapshotInfo.length() > 0); }

  public int getMajorVersion() { return _majorVersion; }
  public int getMinorVersion() { return _minorVersion; }
  public int getPatchLevel() { return _patchLevel; }

  @Override
  public String toString()
  {
    StringBuilder sb = new StringBuilder();
```

```
        sb.append(_majorVersion).append('.');
        sb.append(_minorVersion).append('.');
        sb.append(_patchLevel);
        if (isSnapshot()) {
            sb.append('-').append(_snapshotInfo);
        }
        return sb.toString();
    }

    @Override
    public int hashCode() {
        return _majorVersion + _minorVersion + _patchLevel;
    }

    @Override
    public boolean equals(Object o)
    {
        if (o == this) return true;
        if (o == null) return false;
        if (o.getClass() != getClass()) return false;
        Version other = (Version) o;
        return (other._majorVersion == _majorVersion)
            && (other._minorVersion == _minorVersion)
            && (other._patchLevel == _patchLevel);
    }

    @Override
    public int compareTo(Version other)
    {
        int diff = _majorVersion - other._majorVersion;
        if (diff == 0) {
            diff = _minorVersion - other._minorVersion;
            if (diff == 0) {
                diff = _patchLevel - other._patchLevel;
            }
        }
        return diff;
    }
}
package org.codehaus.jackson;

import java.io.IOException;

import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.type.TypeReference;

/**
* Abstract class that defines the interface that {@link JsonParser} and
```

```
* {@link JsonGenerator} use to serialize and deserialize regular
* Java objects (POJOs aka Beans).
*<p>
* The standard implementation of this class is
* {@link org.codehaus.jackson.map.ObjectMapper}.
*/
public abstract class ObjectCodec
{
  protected ObjectCodec() { }

  /*
  /**********************************************************
  /* API for de-serialization (JSON-to-Object)
  /**********************************************************
   */

  /**
   * Method to deserialize JSON content into a non-container
   * type (it can be an array type, however): typically a bean, array
   * or a wrapper type (like {@link java.lang.Boolean}).
   *<p>
   * Note: this method should NOT be used if the result type is a
   * container ({@link java.util.Collection} or {@link java.util.Map}.
   * The reason is that due to type erasure, key and value types
   * can not be introspected when using this method.
   */
  public abstract <T> T readValue(JsonParser jp, Class<T> valueType)
     throws IOException, JsonProcessingException;

  /**
   * Method to deserialize JSON content into a Java type, reference
   * to which is passed as argument. Type is passed using so-called
   * "super type token" (see )
   * and specifically needs to be used if the root type is a
   * parameterized (generic) container type.
   */
  public abstract <T> T readValue(JsonParser jp, TypeReference<?> valueTypeRef)
     throws IOException, JsonProcessingException;

  /**
   * Method to deserialize JSON content as tree expressed
   * using set of {@link JsonNode} instances. Returns
   * root of the resulting tree (where root can consist
   * of just a single node if the current event is a
   * value event, not container).
   */
  public abstract <T> T readValue(JsonParser jp, JavaType valueType)
     throws IOException, JsonProcessingException;
```

```
/**
 * Method to deserialize JSON content as tree expressed
 * using set of {@link JsonNode} instances. Returns
 * root of the resulting tree (where root can consist
 * of just a single node if the current event is a
 * value event, not container).
 */
public abstract JsonNode readTree(JsonParser jp)
   throws IOException, JsonProcessingException;

/*
/**********************************************************
/* API for serialization (Object-to-JSON)
/**********************************************************
 */

/**
 * Method to serialize given Java Object, using generator
 * provided.
 */
public abstract void writeValue(JsonGenerator jgen, Object value)
   throws IOException, JsonProcessingException;

/**
 * Method to serialize given Json Tree, using generator
 * provided.
 */
public abstract void writeTree(JsonGenerator jgen, JsonNode rootNode)
   throws IOException, JsonProcessingException;

/*
/**********************************************************
/* API for Tree Model handling
/**********************************************************
 */

/**
 * Method for construct root level Object nodes
 * for Tree Model instances.
 *
 * @since 1.2
 */
public abstract JsonNode createObjectNode();

/**
 * Method for construct root level Array nodes
 * for Tree Model instances.
```

```
     *
     * @since 1.2
     */
    public abstract JsonNode createArrayNode();

    /**
     * Method for constructing a {@link JsonParser} for reading
     * contents of a JSON tree, as if it was external serialized
     * JSON content.
     *
     * @since 1.3
     */
    public abstract JsonParser treeAsTokens(JsonNode n);

    /**
     * Convenience method for converting given JSON tree into instance of specified
     * value type. This is equivalent to first constructing a {@link JsonParser} to
     * iterate over contents of the tree, and using that parser for data binding.
     *
     * @since 1.3
     */
    public abstract <T> T treeToValue(JsonNode n, Class<T> valueType)
        throws IOException, JsonProcessingException;
}
package org.codehaus.jackson.io;

public final class NumberOutput
{
    private final static char NULL_CHAR = (char) 0;

    private static int MILLION = 1000000;
    private static int BILLION = 1000000000;
    private static long TEN_BILLION_L = 10000000000L;
    private static long THOUSAND_L = 1000L;

    private static long MIN_INT_AS_LONG = (long) Integer.MIN_VALUE;
    private static long MAX_INT_AS_LONG = (long) Integer.MAX_VALUE;

    final static String SMALLEST_LONG = String.valueOf(Long.MIN_VALUE);

    final static char[] LEADING_TRIPLETS = new char[4000];
    final static char[] FULL_TRIPLETS = new char[4000];
    static {
        /* Let's fill it with NULLs for ignorable leading digits,
         * and digit chars for others
         */
        int ix = 0;
        for (int i1 = 0; i1 < 10; ++i1) {
```

```
        char f1 = (char) ('0' + i1);
        char l1 = (i1 == 0) ? NULL_CHAR : f1;
        for (int i2 = 0; i2 < 10; ++i2) {
            char f2 = (char) ('0' + i2);
            char l2 = (i1 == 0 && i2 == 0) ? NULL_CHAR : f2;
            for (int i3 = 0; i3 < 10; ++i3) {
                // Last is never to be empty
                char f3 = (char) ('0' + i3);
                LEADING_TRIPLETS[ix] = l1;
                LEADING_TRIPLETS[ix+1] = l2;
                LEADING_TRIPLETS[ix+2] = f3;
                FULL_TRIPLETS[ix] = f1;
                FULL_TRIPLETS[ix+1] = f2;
                FULL_TRIPLETS[ix+2] = f3;
                ix += 4;
            }
        }
    }
}

final static byte[] FULL_TRIPLETS_B = new byte[4000];
static {
    for (int i = 0; i < 4000; ++i) {
        FULL_TRIPLETS_B[i] = (byte) FULL_TRIPLETS[i];
    }
}

final static String[] sSmallIntStrs = new String[] {
    "0","1","2","3","4","5","6","7","8","9","10"
};
final static String[] sSmallIntStrs2 = new String[] {
    "-1","-2","-3","-4","-5","-6","-7","-8","-9","-10"
};

/*
/**********************************************************
/* Efficient serialization methods using raw buffers
/**********************************************************
 */

/**
 * @return Offset within buffer after outputting int
 */
public static int outputInt(int value, char[] buffer, int offset)
{
    if (value < 0) {
        if (value == Integer.MIN_VALUE) {
            /* Special case: no matching positive value within range;
```

```
         * let's then "upgrade" to long and output as such.
         */
        return outputLong((long) value, buffer, offset);
    }
    buffer[offset++] = '-';
    value = -value;
}

if (value < MILLION) { // at most 2 triplets...
    if (value < 1000) {
        if (value < 10) {
            buffer[offset++] = (char) ('0' + value);
        } else {
            offset = outputLeadingTriplet(value, buffer, offset);
        }
    } else {
        int thousands = value / 1000;
        value -= (thousands * 1000); // == value % 1000
        offset = outputLeadingTriplet(thousands, buffer, offset);
        offset = outputFullTriplet(value, buffer, offset);
    }
    return offset;
}

// ok, all 3 triplets included
/* Let's first hand possible billions separately before
 * handling 3 triplets. This is possible since we know we
 * can have at most '2' as billion count.
 */
boolean hasBillions = (value >= BILLION);
if (hasBillions) {
    value -= BILLION;
    if (value >= BILLION) {
        value -= BILLION;
        buffer[offset++] = '2';
    } else {
        buffer[offset++] = '1';
    }
}
int newValue = value / 1000;
int ones = (value - (newValue * 1000)); // == value % 1000
value = newValue;
newValue /= 1000;
int thousands = (value - (newValue * 1000));

// value now has millions, which have 1, 2 or 3 digits
if (hasBillions) {
    offset = outputFullTriplet(newValue, buffer, offset);
```

```
    } else {
        offset = outputLeadingTriplet(newValue, buffer, offset);
    }
    offset = outputFullTriplet(thousands, buffer, offset);
    offset = outputFullTriplet(ones, buffer, offset);
    return offset;
}

public static int outputInt(int value, byte[] buffer, int offset)
{
    if (value < 0) {
        if (value == Integer.MIN_VALUE) {
            return outputLong((long) value, buffer, offset);
        }
        buffer[offset++] = '-';
        value = -value;
    }

    if (value < MILLION) { // at most 2 triplets...
        if (value < 1000) {
            if (value < 10) {
                buffer[offset++] = (byte) ('0' + value);
            } else {
                offset = outputLeadingTriplet(value, buffer, offset);
            }
        } else {
            int thousands = value / 1000;
            value -= (thousands * 1000); // == value % 1000
            offset = outputLeadingTriplet(thousands, buffer, offset);
            offset = outputFullTriplet(value, buffer, offset);
        }
        return offset;
    }
    boolean hasBillions = (value >= BILLION);
    if (hasBillions) {
        value -= BILLION;
        if (value >= BILLION) {
            value -= BILLION;
            buffer[offset++] = '2';
        } else {
            buffer[offset++] = '1';
        }
    }
    int newValue = value / 1000;
    int ones = (value - (newValue * 1000)); // == value % 1000
    value = newValue;
    newValue /= 1000;
    int thousands = (value - (newValue * 1000));
```

```
    if (hasBillions) {
        offset = outputFullTriplet(newValue, buffer, offset);
    } else {
        offset = outputLeadingTriplet(newValue, buffer, offset);
    }
    offset = outputFullTriplet(thousands, buffer, offset);
    offset = outputFullTriplet(ones, buffer, offset);
    return offset;
}

/**
 * @return Offset within buffer after outputting int
 */
public static int outputLong(long value, char[] buffer, int offset)
{
    // First: does it actually fit in an int?
    if (value < 0L) {
        /* MIN_INT is actually printed as long, just because its
         * negation is not an int but long
         */
        if (value > MIN_INT_AS_LONG) {
            return outputInt((int) value, buffer, offset);
        }
        if (value == Long.MIN_VALUE) {
            // Special case: no matching positive value within range
            int len = SMALLEST_LONG.length();
            SMALLEST_LONG.getChars(0, len, buffer, offset);
            return (offset + len);
        }
        buffer[offset++] = '-';
        value = -value;
    } else {
        if (value <= MAX_INT_AS_LONG) {
            return outputInt((int) value, buffer, offset);
        }
    }

    /* Ok: real long print. Need to first figure out length
     * in characters, and then print in from end to beginning
     */
    int origOffset = offset;
    offset += calcLongStrLength(value);
    int ptr = offset;

    // First, with long arithmetics:
    while (value > MAX_INT_AS_LONG) { // full triplet
        ptr -= 3;
```

```
        long newValue = value / THOUSAND_L;
        int triplet = (int) (value - newValue * THOUSAND_L);
        outputFullTriplet(triplet, buffer, ptr);
        value = newValue;
    }
    // Then with int arithmetics:
    int ivalue = (int) value;
    while (ivalue >= 1000) { // still full triplet
        ptr -= 3;
        int newValue = ivalue / 1000;
        int triplet = ivalue - (newValue * 1000);
        outputFullTriplet(triplet, buffer, ptr);
        ivalue = newValue;
    }
    // And finally, if anything remains, partial triplet
    outputLeadingTriplet(ivalue, buffer, origOffset);

    return offset;
}

public static int outputLong(long value, byte[] buffer, int offset)
{
    if (value < 0L) {
        if (value > MIN_INT_AS_LONG) {
            return outputInt((int) value, buffer, offset);
        }
        if (value == Long.MIN_VALUE) {
            // Special case: no matching positive value within range
            int len = SMALLEST_LONG.length();
            for (int i = 0; i < len; ++i) {
                buffer[offset++] = (byte) SMALLEST_LONG.charAt(i);
            }
            return offset;
        }
        buffer[offset++] = '-';
        value = -value;
    } else {
        if (value <= MAX_INT_AS_LONG) {
            return outputInt((int) value, buffer, offset);
        }
    }
    int origOffset = offset;
    offset += calcLongStrLength(value);
    int ptr = offset;

    // First, with long arithmetics:
    while (value > MAX_INT_AS_LONG) { // full triplet
        ptr -= 3;
```

```
        long newValue = value / THOUSAND_L;
        int triplet = (int) (value - newValue * THOUSAND_L);
        outputFullTriplet(triplet, buffer, ptr);
        value = newValue;
    }
    // Then with int arithmetics:
    int ivalue = (int) value;
    while (ivalue >= 1000) { // still full triplet
        ptr -= 3;
        int newValue = ivalue / 1000;
        int triplet = ivalue - (newValue * 1000);
        outputFullTriplet(triplet, buffer, ptr);
        ivalue = newValue;
    }
    outputLeadingTriplet(ivalue, buffer, origOffset);
    return offset;
}

/*
/**********************************************************
/* Secondary convenience serialization methods
/**********************************************************
 */

/* !!! 05-Aug-2008, tatus: Any ways to further optimize
 *   these? (or need: only called by diagnostics methods?)
 */

public static String toString(int value)
{
    // Lookup table for small values
    if (value < sSmallIntStrs.length) {
        if (value >= 0) {
            return sSmallIntStrs[value];
        }
        int v2 = -value - 1;
        if (v2 < sSmallIntStrs2.length) {
            return sSmallIntStrs2[v2];
        }
    }
    return Integer.toString(value);
}

public static String toString(long value)
{
    if (value <= Integer.MAX_VALUE &&
        value >= Integer.MIN_VALUE) {
        return toString((int) value);
```

```
    }
    return Long.toString(value);
}

public static String toString(double value)
{
    return Double.toString(value);
}

/*
/**************************************************************
/* Internal methods
/**************************************************************
 */

private static int outputLeadingTriplet(int triplet, char[] buffer, int offset)
{
    int digitOffset = (triplet << 2);
    char c = LEADING_TRIPLETS[digitOffset++];
    if (c != NULL_CHAR) {
        buffer[offset++] = c;
    }
    c = LEADING_TRIPLETS[digitOffset++];
    if (c != NULL_CHAR) {
        buffer[offset++] = c;
    }
    // Last is required to be non-empty
    buffer[offset++] = LEADING_TRIPLETS[digitOffset];
    return offset;
}

private static int outputLeadingTriplet(int triplet, byte[] buffer, int offset)
{
    int digitOffset = (triplet << 2);
    char c = LEADING_TRIPLETS[digitOffset++];
    if (c != NULL_CHAR) {
        buffer[offset++] = (byte) c;
    }
    c = LEADING_TRIPLETS[digitOffset++];
    if (c != NULL_CHAR) {
        buffer[offset++] = (byte) c;
    }
    // Last is required to be non-empty
    buffer[offset++] = (byte) LEADING_TRIPLETS[digitOffset];
    return offset;
}

private static int outputFullTriplet(int triplet, char[] buffer, int offset)
```

```
    {
        int digitOffset = (triplet << 2);
        buffer[offset++] = FULL_TRIPLETS[digitOffset++];
        buffer[offset++] = FULL_TRIPLETS[digitOffset++];
        buffer[offset++] = FULL_TRIPLETS[digitOffset];
        return offset;
    }

    private static int outputFullTriplet(int triplet, byte[] buffer, int offset)
    {
        int digitOffset = (triplet << 2);
        buffer[offset++] = FULL_TRIPLETS_B[digitOffset++];
        buffer[offset++] = FULL_TRIPLETS_B[digitOffset++];
        buffer[offset++] = FULL_TRIPLETS_B[digitOffset];
        return offset;
    }

    /**
     *<p>
     * Pre-conditions: posValue is positive, and larger than
     * Integer.MAX_VALUE (about 2 billions).
     */
    private static int calcLongStrLength(long posValue)
    {
        int len = 10;
        long comp = TEN_BILLION_L;

        // 19 is longest, need to worry about overflow
        while (posValue >= comp) {
            if (len == 19) {
                break;
            }
            ++len;
            comp = (comp << 3) + (comp << 1); // 10x
        }
        return len;
    }
}
package org.codehaus.jackson.io;

import java.io.*;

import org.codehaus.jackson.util.BufferRecycler;
import org.codehaus.jackson.util.TextBuffer;

/**
* Efficient alternative to {@link StringWriter}, based on using segmented
* internal buffer. Initial input buffer is also recyclable.
```

```
*<p>
* This class is most useful when serializing JSON content as a String:
* if so, instance of this class can be given as the writer to
* <code>JsonGenerator</code>.
*
* @since 1.3
*/
public final class SegmentedStringWriter
  extends Writer
{
  final protected TextBuffer _buffer;

  public SegmentedStringWriter(BufferRecycler br)
  {
     super();
     _buffer = new TextBuffer(br);
  }

  /*
  /**********************************************************
  /* java.io.Writer implementation
  /**********************************************************
   */

  @Override
  public Writer append(char c)
  {
     write(c);
     return this;
  }

  @Override
  public Writer append(CharSequence csq)
  {
 String str = csq.toString();
 _buffer.append(str, 0, str.length());
     return this;
  }

  @Override
  public Writer append(CharSequence csq, int start, int end)
  {
 String str = csq.subSequence(start, end).toString();
 _buffer.append(str, 0, str.length());
     return this;
  }

  @Override public void close() { } // NOP
```

```
  @Override public void flush() { } // NOP

  @Override
  public void write(char[] cbuf) {
     _buffer.append(cbuf, 0, cbuf.length);
  }

  @Override
  public void write(char[] cbuf, int off, int len) {
     _buffer.append(cbuf, off, len);
  }

  @Override
  public void write(int c) {
     _buffer.append((char) c);
  }

  @Override
  public void write(String str) { _buffer.append(str, 0, str.length()); }

  @Override
  public void write(String str, int off, int len) {
     _buffer.append(str, 0, str.length());
  }

  /*
  /**********************************************************
  /* Extended API
  /**********************************************************
   */

  /**
   * Main access method that will construct a String that contains
   * all the contents, release all internal buffers we may have,
   * and return result String.
   * Note that the method is not idempotent -- if called second time,
   * will just return an empty String.
   */
  public String getAndClear()
  {
     String result = _buffer.contentsAsString();
     _buffer.releaseBuffers();
     return result;
  }
}

package org.codehaus.jackson.io;
```

```
import java.io.*;


/**
* Simple basic class for optimized readers in this package; implements
* "cookie-cutter" methods that are used by all actual implementations.
*/
abstract class BaseReader
  extends Reader
{
  /**
   * JSON actually limits available Unicode range in the high end
   * to the same as xml (to basically limit UTF-8 max byte sequence
   * length to 4)
   */
  final protected static int LAST_VALID_UNICODE_CHAR = 0x10FFFF;

  final protected static char NULL_CHAR = (char) 0;
  final protected static char NULL_BYTE = (byte) 0;

  final protected IOContext _context;

  protected InputStream _in;

  protected byte[] _buffer;

  protected int _ptr;
  protected int _length;

  /*
  /**********************************************************
  /* Life-cycle
  /**********************************************************
   */

  protected BaseReader(IOContext context,
                       InputStream in, byte[] buf, int ptr, int len)
  {
     _context = context;
     _in = in;
     _buffer = buf;
     _ptr = ptr;
     _length = len;
  }

  /*
  /**********************************************************
```

```
/* Reader API
/**********************************************************
 */

@Override
public void close() throws IOException
{
   InputStream in = _in;

   if (in != null) {
      _in = null;
      freeBuffers();
      in.close();
   }
}

protected char[] _tmpBuf = null;

/**
 * Although this method is implemented by the base class, AND it should
 * never be called by main code, let's still implement it bit more
 * efficiently just in case
 */
@Override
public int read() throws IOException
{
   if (_tmpBuf == null) {
      _tmpBuf = new char[1];
   }
   if (read(_tmpBuf, 0, 1) < 1) {
      return -1;
   }
   return _tmpBuf[0];
}

/*
/**********************************************************
/* Internal/package methods:
/**********************************************************
 */

/**
 * This method should be called along with (or instead of) normal
 * close. After calling this method, no further reads should be tried.
 * Method will try to recycle read buffers (if any).
 */
public final void freeBuffers()
{
```

```
        byte[] buf = _buffer;
        if (buf != null) {
            _buffer = null;
            _context.releaseReadIOBuffer(buf);
        }
    }

    protected void reportBounds(char[] cbuf, int start, int len)
        throws IOException
    {
        throw new ArrayIndexOutOfBoundsException("read(buf,"+start+","+len+"), cbuf["+cbuf.length+"]");
    }

    protected void reportStrangeStream()
        throws IOException
    {
        throw new IOException("Strange I/O stream, returned 0 bytes on read");
    }
}
package org.codehaus.jackson.io;

import java.io.*;


/**
* Since JDK does not come with UTF-32/UCS-4, let's implement a simple
* decoder to use.
*/
public final class UTF32Reader
    extends BaseReader
{
    final boolean mBigEndian;

    /**
     * Although input is fine with full Unicode set, Java still uses
     * 16-bit chars, so we may have to split high-order chars into
     * surrogate pairs.
     */
    char mSurrogate = NULL_CHAR;

    /**
     * Total read character count; used for error reporting purposes
     */
    int mCharCount = 0;

    /**
     * Total read byte count; used for error reporting purposes
     */
```

```
    int mByteCount = 0;

    /*
    ////////////////////////////////////////
    // Life-cycle
    ////////////////////////////////////////
    */

    public UTF32Reader(IOContext ctxt,
                       InputStream in, byte[] buf, int ptr, int len,
                       boolean isBigEndian)
    {
        super(ctxt, in, buf, ptr, len);
        mBigEndian = isBigEndian;
    }

    /*
    ////////////////////////////////////////
    // Public API
    ////////////////////////////////////////
    */

    @Override
    public int read(char[] cbuf, int start, int len)
        throws IOException
    {
        // Already EOF?
        if (_buffer == null) {
            return -1;
        }
        if (len < 1) {
            return len;
        }
        // Let's then ensure there's enough room...
        if (start < 0 || (start+len) > cbuf.length) {
            reportBounds(cbuf, start, len);
        }

        len += start;
        int outPtr = start;

        // Ok, first; do we have a surrogate from last round?
        if (mSurrogate != NULL_CHAR) {
            cbuf[outPtr++] = mSurrogate;
            mSurrogate = NULL_CHAR;
            // No need to load more, already got one char
        } else {
            /* Note: we'll try to avoid blocking as much as possible. As a
```

```
     * result, we only need to get 4 bytes for a full char.
     */
    int left = (_length - _ptr);
    if (left < 4) {
        if (!loadMore(left)) { // (legal) EOF?
            return -1;
        }
    }
}

main_loop:
while (outPtr < len) {
    int ptr = _ptr;
    int ch;

    if (mBigEndian) {
        ch = (_buffer[ptr] << 24) | ((_buffer[ptr+1] & 0xFF) << 16)
            | ((_buffer[ptr+2] & 0xFF) << 8) | (_buffer[ptr+3] & 0xFF);
    } else {
        ch = (_buffer[ptr] & 0xFF) | ((_buffer[ptr+1] & 0xFF) << 8)
            | ((_buffer[ptr+2] & 0xFF) << 16) | (_buffer[ptr+3] << 24);
    }
    _ptr += 4;

    // Does it need to be split to surrogates?
    // (also, we can and need to verify illegal chars)
    if (ch > 0xFFFF) { // need to split into surrogates?
        if (ch > LAST_VALID_UNICODE_CHAR) {
            reportInvalid(ch, outPtr-start,
                    "(above "+Integer.toHexString(LAST_VALID_UNICODE_CHAR)+") ");
        }
        ch -= 0x10000; // to normalize it starting with 0x0
        cbuf[outPtr++] = (char) (0xD800 + (ch >> 10));
        // hmmh. can this ever be 0? (not legal, at least?)
        ch = (0xDC00 | (ch & 0x03FF));
        // Room for second part?
        if (outPtr >= len) { // nope
            mSurrogate = (char) ch;
            break main_loop;
        }
    }
    cbuf[outPtr++] = (char) ch;
    if (_ptr >= _length) {
        break main_loop;
    }
}

len = outPtr - start;
```

```
    mCharCount += len;
    return len;
}

/*
//////////////////////////////////////
// Internal methods
//////////////////////////////////////
*/

private void reportUnexpectedEOF(int gotBytes, int needed)
    throws IOException
{
    int bytePos = mByteCount + gotBytes;
    int charPos = mCharCount;

    throw new CharConversionException("Unexpected EOF in the middle of a 4-byte UTF-32 char: got "
                                      +gotBytes+", needed "+needed
                                      +", at char #"+charPos+", byte #"+bytePos+")");
}

private void reportInvalid(int value, int offset, String msg)
    throws IOException
{
    int bytePos = mByteCount + _ptr - 1;
    int charPos = mCharCount + offset;

    throw new CharConversionException("Invalid UTF-32 character 0x"
                                      +Integer.toHexString(value)
                                      +msg+" at char #"+charPos+", byte #"+bytePos+")");
}

/**
 * @param available Number of "unused" bytes in the input buffer
 *
 * @return True, if enough bytes were read to allow decoding of at least
 *   one full character; false if EOF was encountered instead.
 */
private boolean loadMore(int available)
    throws IOException
{
    mByteCount += (_length - available);

    // Bytes that need to be moved to the beginning of buffer?
    if (available > 0) {
        if (_ptr > 0) {
            for (int i = 0; i < available; ++i) {
                _buffer[i] = _buffer[_ptr+i];
```

```
            }
            _ptr = 0;
        }
        _length = available;
    } else {
        /* Ok; here we can actually reasonably expect an EOF,
         * so let's do a separate read right away:
         */
        _ptr = 0;
        int count = _in.read(_buffer);
        if (count < 1) {
            _length = 0;
            if (count < 0) { // -1
                freeBuffers(); // to help GC?
                return false;
            }
            // 0 count is no good; let's err out
            reportStrangeStream();
        }
        _length = count;
    }

    /* Need at least 4 bytes; if we don't get that many, it's an
     * error.
     */
    while (_length < 4) {
        int count = _in.read(_buffer, _length, _buffer.length - _length);
        if (count < 1) {
            if (count < 0) { // -1, EOF... no good!
                freeBuffers(); // to help GC?
                reportUnexpectedEOF(_length, 4);
            }
            // 0 count is no good; let's err out
            reportStrangeStream();
        }
        _length += count;
    }
    return true;
  }
}
<body>
This package contains I/O helper classes Jackson itself uses, but that
are not exposed for external reuse.
</body>
package org.codehaus.jackson.io;

import org.codehaus.jackson.SerializableString;
```

```
/**
* String token that can lazily serialize String contained and then reuse that
* serialization later on. This is similar to JDBC prepared statements, for example,
* in that instances should only be created when they are used more than use;
* prime candidates are various serializers.
*<p>
* Class is final for performance reasons and since this is not designed to
* be extensible or customizable (customizations would occur in calling code)
*
* @since 1.6
*/
public class SerializedString implements SerializableString
{
  protected final String _value;

  /* 13-Dec-2010, tatu: Whether use volatile or not is actually an important
   *   decision for multi-core use cases. Cost of volatility can be non-trivial
   *   for heavy use cases, and serialized-string instances are accessed often.
   *   Given that all codepaths with common Jackson usage patterns go through
   *   a few memory barries (mostly with cache/reuse pool access) it seems safe
   *   enough to omit volatiles here, given how simple lazy initialization is.
   *   This can be compared to how {@link String#intern} works; lazily and
   *   without synchronization or use of volatile keyword.
   */

  protected /*volatile*/ byte[] _quotedUTF8Ref;

  protected /*volatile*/ byte[] _unquotedUTF8Ref;

  protected /*volatile*/ char[] _quotedChars;

  public SerializedString(String v) { _value = v; }

  /*
  /**********************************************************
  /* API
  /**********************************************************
   */

  public final String getValue() { return _value; }

  /**
   * Returns length of the String as characters
   */
  public final int charLength() { return _value.length(); }

  public final char[] asQuotedChars()
  {
```

```
    char[] result = _quotedChars;
    if (result == null) {
        result = JsonStringEncoder.getInstance().quoteAsString(_value);
        _quotedChars = result;
    }
    return result;
}

/**
 * Accessor for accessing value that has been quoted using JSON
 * quoting rules, and encoded using UTF-8 encoding.
 */
public final byte[] asUnquotedUTF8()
{
    byte[] result = _unquotedUTF8Ref;
    if (result == null) {
        result = JsonStringEncoder.getInstance().encodeAsUTF8(_value);
        _unquotedUTF8Ref  = result;
    }
    return result;
}

/**
 * Accessor for accessing value as is (without JSON quoting)
 * encoded using UTF-8 encoding.
 */
public final byte[] asQuotedUTF8()
{
    byte[] result = _quotedUTF8Ref;
    if (result == null) {
        result = JsonStringEncoder.getInstance().quoteAsUTF8(_value);
        _quotedUTF8Ref = result;
    }
    return result;
}

/*
/**********************************************************
/* Standard method overrides
/**********************************************************
 */

@Override
public final String toString() { return _value; }

@Override
public final int hashCode() { return _value.hashCode(); }
```

```
  @Override
  public final boolean equals(Object o)
  {
     if (o == this) return true;
     if (o == null || o.getClass() != getClass()) return false;
     SerializedString other = (SerializedString) o;
     return _value.equals(other._value);
  }
}
package org.codehaus.jackson.io;

public final class NumberInput
{
  /**
   * Textual representation of a double constant that can cause nasty problems
   * with JDK (see http://www.exploringbinary.com/java-hangs-when-converting-2-2250738585072012e-308).
   */
  public final static String NASTY_SMALL_DOUBLE = "2.2250738585072012e-308";

  /**
   * Constants needed for parsing longs from basic int parsing methods
   */
  final static long L_BILLION = 1000000000;

  final static String MIN_LONG_STR_NO_SIGN = String.valueOf(Long.MIN_VALUE).substring(1);
  final static String MAX_LONG_STR = String.valueOf(Long.MAX_VALUE);

  /**
   * Fast method for parsing integers that are known to fit into
   * regular 32-bit signed int type. This means that length is
   * between 1 and 9 digits (inclusive)
   *<p>
   * Note: public to let unit tests call it
   */
  public final static int parseInt(char[] digitChars, int offset, int len)
  {
     int num = digitChars[offset] - '0';
     len += offset;
     // This looks ugly, but appears the fastest way (as per measurements)
     if (++offset < len) {
        num = (num * 10) + (digitChars[offset] - '0');
        if (++offset < len) {
           num = (num * 10) + (digitChars[offset] - '0');
           if (++offset < len) {
              num = (num * 10) + (digitChars[offset] - '0');
              if (++offset < len) {
                 num = (num * 10) + (digitChars[offset] - '0');
                 if (++offset < len) {
```

```
                num = (num * 10) + (digitChars[offset] - '0');
                if (++offset < len) {
                    num = (num * 10) + (digitChars[offset] - '0');
                    if (++offset < len) {
                        num = (num * 10) + (digitChars[offset] - '0');
                        if (++offset < len) {
                            num = (num * 10) + (digitChars[offset] - '0');
                        }
                    }
                }
            }
        }
    }
    return num;
}

/**
 * Helper method to (more) efficiently parse integer numbers from
 * String values.
 *
 * @since 1.7
 */
public final static int parseInt(String str)
{
    /* Ok: let's keep strategy simple: ignoring optional minus sign,
     * we'll accept 1 - 9 digits and parse things efficiently;
     * otherwise just defer to JDK parse functionality.
     */
    char c = str.charAt(0);
    int length = str.length();
    boolean negative = (c == '-');
    int offset = 1;
    // must have 1 - 9 digits after optional sign:
    // negative?
    if (negative) {
        if (length == 1 || length > 10) {
            return Integer.parseInt(str);
        }
        c = str.charAt(offset++);
    } else {
        if (length > 9) {
            return Integer.parseInt(str);
        }
    }
    if (c > '9' || c < '0') {
        return Integer.parseInt(str);
```

```
    }
    int num = c - '0';
    if (offset < length) {
        c = str.charAt(offset++);
        if (c > '9' || c < '0') {
            return Integer.parseInt(str);
        }
        num = (num * 10) + (c - '0');
        if (offset < length) {
            c = str.charAt(offset++);
            if (c > '9' || c < '0') {
                return Integer.parseInt(str);
            }
            num = (num * 10) + (c - '0');
            // Let's just loop if we have more than 3 digits:
            if (offset < length) {
                do {
                    c = str.charAt(offset++);
                    if (c > '9' || c < '0') {
                        return Integer.parseInt(str);
                    }
                    num = (num * 10) + (c - '0');
                } while (offset < length);
            }
        }
    }
    return negative ? -num : num;
}

public final static long parseLong(char[] digitChars, int offset, int len)
{
    // Note: caller must ensure length is [10, 18]
    int len1 = len-9;
    long val = parseInt(digitChars, offset, len1) * L_BILLION;
    return val + (long) parseInt(digitChars, offset+len1, 9);
}

public final static long parseLong(String str)
{
    /* Ok, now; as the very first thing, let's just optimize case of "fake longs";
     * that is, if we know they must be ints, call int parsing
     */
    int length = str.length();
    if (length <= 9) {
        return (long) parseInt(str);
    }
    // !!! TODO: implement efficient 2-int parsing...
    return Long.parseLong(str);
```

```
}

/**
 * Helper method for determining if given String representation of
 * an integral number would fit in 64-bit Java long or not.
 * Note that input String must NOT contain leading minus sign (even
 * if 'negative' is set to true).
 *
 * @param negative Whether original number had a minus sign (which is
 *    NOT passed to this method) or not
 */
public final static boolean inLongRange(char[] digitChars, int offset, int len,
       boolean negative)
{
   String cmpStr = negative ? MIN_LONG_STR_NO_SIGN : MAX_LONG_STR;
   int cmpLen = cmpStr.length();
   if (len < cmpLen) return true;
   if (len > cmpLen) return false;

   for (int i = 0; i < cmpLen; ++i) {
      int diff = digitChars[offset+i] - cmpStr.charAt(i);
      if (diff != 0) {
         return (diff < 0);
      }
   }
   return true;
}

/**
 * Similar to {@link #inLongRange(char[],int,int,boolean)}, but
 * with String argument
 *
 * @param negative Whether original number had a minus sign (which is
 *    NOT passed to this method) or not
 *
 * @since 1.5.0
 */
public final static boolean inLongRange(String numberStr, boolean negative)
{
   String cmpStr = negative ? MIN_LONG_STR_NO_SIGN : MAX_LONG_STR;
   int cmpLen = cmpStr.length();
   int actualLen = numberStr.length();
   if (actualLen < cmpLen) return true;
   if (actualLen > cmpLen) return false;

   // could perhaps just use String.compareTo()?
   for (int i = 0; i < cmpLen; ++i) {
      int diff = numberStr.charAt(i) - cmpStr.charAt(i);
```

```
            if (diff != 0) {
                return (diff < 0);
            }
        }
        return true;
    }

    /**
     * @since 1.6
     */
    public static int parseAsInt(String input, int defaultValue)
    {
        if (input == null) {
            return defaultValue;
        }
        input = input.trim();
        int len = input.length();
        if (len == 0) {
            return defaultValue;
        }
        // One more thing: use integer parsing for 'simple'
        int i = 0;
        if (i < len) { // skip leading sign:
            char c = input.charAt(0);
            if (c == '+') { // for plus, actually physically remove
                input = input.substring(1);
                len = input.length();
            } else if (c == '-') { // minus, just skip for checks, must retain
                ++i;
            }
        }
        for (; i < len; ++i) {
            char c = input.charAt(i);
            // if other symbols, parse as Double, coerce
            if (c > '9' || c < '0') {
                try {
                    return (int) parseDouble(input);
                } catch (NumberFormatException e) {
                    return defaultValue;
                }
            }
        }
        try {
            return Integer.parseInt(input);
        } catch (NumberFormatException e) { }
        return defaultValue;
    }
```

```
/**
 * @since 1.6
 */
public static long parseAsLong(String input, long defaultValue)
{
    if (input == null) {
        return defaultValue;
    }
    input = input.trim();
    int len = input.length();
    if (len == 0) {
        return defaultValue;
    }
    // One more thing: use integer parsing for 'simple'
    int i = 0;
    if (i < len) { // skip leading sign:
        char c = input.charAt(0);
        if (c == '+') { // for plus, actually physically remove
            input = input.substring(1);
            len = input.length();
        } else if (c == '-') { // minus, just skip for checks, must retain
            ++i;
        }
    }
    for (; i < len; ++i) {
        char c = input.charAt(i);
        // if other symbols, parse as Double, coerce
        if (c > '9' || c < '0') {
            try {
                return (long) parseDouble(input);
            } catch (NumberFormatException e) {
                return defaultValue;
            }
        }
    }
    try {
        return Long.parseLong(input);
    } catch (NumberFormatException e) { }
    return defaultValue;
}

/**
 * @since 1.6
 */
public static double parseAsDouble(String input, double defaultValue)
{
    if (input == null) {
        return defaultValue;
```

```
        }
        input = input.trim();
        int len = input.length();
        if (len == 0) {
            return defaultValue;
        }
        try {
            return parseDouble(input);
        } catch (NumberFormatException e) { }
        return defaultValue;
    }

    /**
     * @since 1.8
     */
    public final static double parseDouble(String numStr) throws NumberFormatException
    {
        // [JACKSON-486]: avoid some nasty float representations... but should it be MIN_NORMAL or
MIN_VALUE?
        if (NASTY_SMALL_DOUBLE.equals(numStr)) {
            return Double.MIN_NORMAL;
        }
        return Double.parseDouble(numStr);
    }
}
package org.codehaus.jackson.io;

import java.lang.ref.SoftReference;

import org.codehaus.jackson.util.BufferRecycler;
import org.codehaus.jackson.util.ByteArrayBuilder;
import org.codehaus.jackson.util.CharTypes;
import org.codehaus.jackson.util.TextBuffer;

/**
* Helper class used for efficient encoding of JSON String values (including
* JSON field names) into Strings or UTF-8 byte arrays.
*<p>
* Note that methods in here are somewhat optimized, but not ridiculously so.
* Reason is that conversion method results are expected to be cached so that
* these methods will not be hot spots during normal operation.
*
* @since 1.6
*/
public final class JsonStringEncoder
{
    private final static char[] HEX_CHARS = CharTypes.copyHexChars();
```

```
private final static byte[] HEX_BYTES = CharTypes.copyHexBytes();

private final static int SURR1_FIRST = 0xD800;
private final static int SURR1_LAST = 0xDBFF;
private final static int SURR2_FIRST = 0xDC00;
private final static int SURR2_LAST = 0xDFFF;

private final static int INT_BACKSLASH = '\\';
private final static int INT_U = 'u';
private final static int INT_0 = '0';

/**
 * This <code>ThreadLocal</code> contains a {@link java.lang.ref.SoftRerefence}
 * to a {@link BufferRecycler} used to provide a low-cost
 * buffer recycling between reader and writer instances.
 */
final protected static ThreadLocal<SoftReference<JsonStringEncoder>> _threadEncoder
   = new ThreadLocal<SoftReference<JsonStringEncoder>>();

/**
 * Lazily constructed text buffer used to produce JSON encoded Strings
 * as characters (without UTF-8 encoding)
 */
protected TextBuffer _textBuffer;

/**
 * Lazily-constructed builder used for UTF-8 encoding of text values
 * (quoted and unquoted)
 */
protected ByteArrayBuilder _byteBuilder;

/**
 * Temporary buffer used for composing quote/escape sequences
 */
protected final char[] _quoteBuffer;

/*
/**********************************************************
/* Construction, instance access
/**********************************************************
 */

public JsonStringEncoder()
{
   _quoteBuffer = new char[6];
   _quoteBuffer[0] = '\\';
   _quoteBuffer[2] = '0';
   _quoteBuffer[3] = '0';
```

```
}

/**
 * Factory method for getting an instance; this is either recycled per-thread instance,
 * or a newly constructed one.
 */
public static JsonStringEncoder getInstance()
{
    SoftReference<JsonStringEncoder> ref = _threadEncoder.get();
    JsonStringEncoder enc = (ref == null) ? null : ref.get();

    if (enc == null) {
        enc = new JsonStringEncoder();
        _threadEncoder.set(new SoftReference<JsonStringEncoder>(enc));
    }
    return enc;
}

/*
/**********************************************************
/* Public API
/**********************************************************
 */

/**
 * Method that will quote text contents using JSON standard quoting,
 * and return results as a character array
 */
public char[] quoteAsString(String input)
{
    TextBuffer textBuffer = _textBuffer;
    if (textBuffer == null) {
        // no allocator; can add if we must, shouldn't need to
        _textBuffer = textBuffer = new TextBuffer(null);
    }
    char[] outputBuffer = textBuffer.emptyAndGetCurrentSegment();
    final int[] escCodes = CharTypes.getOutputEscapes();
    final int escCodeCount = escCodes.length;
    int inPtr = 0;
    final int inputLen = input.length();
    int outPtr = 0;

    outer_loop:
    while (inPtr < inputLen) {
        tight_loop:
        while (true) {
            char c = input.charAt(inPtr);
            if (c < escCodeCount && escCodes[c] != 0) {
```

```
                break tight_loop;
            }
            if (outPtr >= outputBuffer.length) {
                outputBuffer = textBuffer.finishCurrentSegment();
                outPtr = 0;
            }
            outputBuffer[outPtr++] = c;
            if (++inPtr >= inputLen) {
                break outer_loop;
            }
        }
        // something to escape; 2 or 6-char variant?
        int escCode = escCodes[input.charAt(inPtr++)];
        int length = _appendSingleEscape(escCode, _quoteBuffer);
        if ((outPtr + length) > outputBuffer.length) {
            int first = outputBuffer.length - outPtr;
            if (first > 0) {
                System.arraycopy(_quoteBuffer, 0, outputBuffer, outPtr, first);
            }
            outputBuffer = textBuffer.finishCurrentSegment();
            int second = length - first;
            System.arraycopy(_quoteBuffer, first, outputBuffer, outPtr, second);
            outPtr += second;
        } else {
            System.arraycopy(_quoteBuffer, 0, outputBuffer, outPtr, length);
            outPtr += length;
        }

    }
    textBuffer.setCurrentLength(outPtr);
    return textBuffer.contentsAsArray();
}

/**
 * Will quote given JSON String value using standard quoting, encode
 * results as UTF-8, and return result as a byte array.
 */
public byte[] quoteAsUTF8(String text)
{
    ByteArrayBuilder byteBuilder = _byteBuilder;
    if (byteBuilder == null) {
        // no allocator; can add if we must, shouldn't need to
        _byteBuilder = byteBuilder = new ByteArrayBuilder(null);
    }
    int inputPtr = 0;
    int inputEnd = text.length();
    int outputPtr = 0;
    byte[] outputBuffer = byteBuilder.resetAndGetFirstSegment();
```

```
main_loop:
while (inputPtr < inputEnd) {
   final int[] escCodes = CharTypes.getOutputEscapes();

   inner_loop: // ascii and escapes
   while (true) {
      int ch = text.charAt(inputPtr);
      if (ch > 0x7F || escCodes[ch] != 0) {
         break inner_loop;
      }
      if (outputPtr >= outputBuffer.length) {
         outputBuffer = byteBuilder.finishCurrentSegment();
         outputPtr = 0;
      }
      outputBuffer[outputPtr++] = (byte) ch;
      if (++inputPtr >= inputEnd) {
         break main_loop;
      }
   }
   if (outputPtr >= outputBuffer.length) {
      outputBuffer = byteBuilder.finishCurrentSegment();
      outputPtr = 0;
   }
   // Ok, so what did we hit?
   int ch = (int) text.charAt(inputPtr++);
   if (ch <= 0x7F) { // needs quoting
      int escape = escCodes[ch];
      // ctrl-char, 6-byte escape...
      outputPtr = _appendByteEscape(escape, byteBuilder, outputPtr);
      outputBuffer = byteBuilder.getCurrentSegment();
      continue main_loop;
   } else if (ch <= 0x7FF) { // fine, just needs 2 byte output
      outputBuffer[outputPtr++] = (byte) (0xc0 | (ch >> 6));
      ch = (0x80 | (ch & 0x3f));
   } else { // 3 or 4 bytes
      // Surrogates?
      if (ch < SURR1_FIRST || ch > SURR2_LAST) { // nope
         outputBuffer[outputPtr++] = (byte) (0xe0 | (ch >> 12));
         if (outputPtr >= outputBuffer.length) {
            outputBuffer = byteBuilder.finishCurrentSegment();
            outputPtr = 0;
         }
         outputBuffer[outputPtr++] = (byte) (0x80 | ((ch >> 6) & 0x3f));
         ch = (0x80 | (ch & 0x3f));
      } else { // yes, surrogate pair
         if (ch > SURR1_LAST) { // must be from first range
            _throwIllegalSurrogate(ch);
```

```
                }
                // and if so, followed by another from next range
                if (inputPtr >= inputEnd) {
                    _throwIllegalSurrogate(ch);
                }
                ch = _convertSurrogate(ch, text.charAt(inputPtr++));
                if (ch > 0x10FFFF) { // illegal, as per RFC 4627
                    _throwIllegalSurrogate(ch);
                }
                outputBuffer[outputPtr++] = (byte) (0xf0 | (ch >> 18));
                if (outputPtr >= outputBuffer.length) {
                    outputBuffer = byteBuilder.finishCurrentSegment();
                    outputPtr = 0;
                }
                outputBuffer[outputPtr++] = (byte) (0x80 | ((ch >> 12) & 0x3f));
                if (outputPtr >= outputBuffer.length) {
                    outputBuffer = byteBuilder.finishCurrentSegment();
                    outputPtr = 0;
                }
                outputBuffer[outputPtr++] = (byte) (0x80 | ((ch >> 6) & 0x3f));
                ch = (0x80 | (ch & 0x3f));
            }
        }
        if (outputPtr >= outputBuffer.length) {
            outputBuffer = byteBuilder.finishCurrentSegment();
            outputPtr = 0;
        }
        outputBuffer[outputPtr++] = (byte) ch;
    }
    return _byteBuilder.completeAndCoalesce(outputPtr);
}

/**
 * Will encode given String as UTF-8 (without any quoting), return
 * resulting byte array.
 */
public byte[] encodeAsUTF8(String text)
{
    ByteArrayBuilder byteBuilder = _byteBuilder;
    if (byteBuilder == null) {
        // no allocator; can add if we must, shouldn't need to
        _byteBuilder = byteBuilder = new ByteArrayBuilder(null);
    }
    int inputPtr = 0;
    int inputEnd = text.length();
    int outputPtr = 0;
    byte[] outputBuffer = byteBuilder.resetAndGetFirstSegment();
    int outputEnd = outputBuffer.length;
```

```
main_loop:
while (inputPtr < inputEnd) {
   int c = text.charAt(inputPtr++);

   // first tight loop for ascii
   while (c <= 0x7F) {
      if (outputPtr >= outputEnd) {
         outputBuffer = byteBuilder.finishCurrentSegment();
         outputEnd = outputBuffer.length;
         outputPtr = 0;
      }
      outputBuffer[outputPtr++] = (byte) c;
      if (inputPtr >= inputEnd) {
         break main_loop;
      }
      c = text.charAt(inputPtr++);
   }

   // then multi-byte...
   if (outputPtr >= outputEnd) {
      outputBuffer = byteBuilder.finishCurrentSegment();
      outputEnd = outputBuffer.length;
      outputPtr = 0;
   }
   if (c < 0x800) { // 2-byte
      outputBuffer[outputPtr++] = (byte) (0xc0 | (c >> 6));
   } else { // 3 or 4 bytes
      // Surrogates?
      if (c < SURR1_FIRST || c > SURR2_LAST) { // nope
         outputBuffer[outputPtr++] = (byte) (0xe0 | (c >> 12));
         if (outputPtr >= outputEnd) {
            outputBuffer = byteBuilder.finishCurrentSegment();
            outputEnd = outputBuffer.length;
            outputPtr = 0;
         }
         outputBuffer[outputPtr++] = (byte) (0x80 | ((c >> 6) & 0x3f));
      } else { // yes, surrogate pair
         if (c > SURR1_LAST) { // must be from first range
            _throwIllegalSurrogate(c);
         }
         // and if so, followed by another from next range
         if (inputPtr >= inputEnd) {
            _throwIllegalSurrogate(c);
         }
         c = _convertSurrogate(c, text.charAt(inputPtr++));
         if (c > 0x10FFFF) { // illegal, as per RFC 4627
            _throwIllegalSurrogate(c);
```

```
            }
            outputBuffer[outputPtr++] = (byte) (0xf0 | (c >> 18));
            if (outputPtr >= outputEnd) {
               outputBuffer = byteBuilder.finishCurrentSegment();
               outputEnd = outputBuffer.length;
               outputPtr = 0;
            }
            outputBuffer[outputPtr++] = (byte) (0x80 | ((c >> 12) & 0x3f));
            if (outputPtr >= outputEnd) {
               outputBuffer = byteBuilder.finishCurrentSegment();
               outputEnd = outputBuffer.length;
               outputPtr = 0;
            }
            outputBuffer[outputPtr++] = (byte) (0x80 | ((c >> 6) & 0x3f));
         }
      }
      if (outputPtr >= outputEnd) {
         outputBuffer = byteBuilder.finishCurrentSegment();
         outputEnd = outputBuffer.length;
         outputPtr = 0;
      }
      outputBuffer[outputPtr++] = (byte) (0x80 | (c & 0x3f));
   }
   return _byteBuilder.completeAndCoalesce(outputPtr);
}

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

private int _appendSingleEscape(int escCode, char[] quoteBuffer)
{
   if (escCode < 0) { // control char, value -(char + 1)
      int value = -(escCode + 1);
      quoteBuffer[1] = 'u';
      // We know it's a control char, so only the last 2 chars are non-0
      quoteBuffer[4] = HEX_CHARS[value >> 4];
      quoteBuffer[5] = HEX_CHARS[value & 0xF];
      return 6;
   }
   quoteBuffer[1] = (char) escCode;
   return 2;
}

private int _appendByteEscape(int escCode, ByteArrayBuilder byteBuilder, int ptr)
{
```

```
            byteBuilder.setCurrentSegmentLength(ptr);
            byteBuilder.append(INT_BACKSLASH);
            if (escCode < 0) { // control char, value -(char + 1)
                int value = -(escCode + 1);
                byteBuilder.append(INT_U);
                byteBuilder.append(INT_0);
                byteBuilder.append(INT_0);
                // We know it's a control char, so only the last 2 chars are non-0
                byteBuilder.append(HEX_BYTES[value >> 4]);
                byteBuilder.append(HEX_BYTES[value & 0xF]);
            } else {
                byteBuilder.append((byte) escCode);
            }
            return byteBuilder.getCurrentSegmentLength();
        }

        /**
         * Method called to calculate UTF codepoint, from a surrogate pair.
         */
        private int _convertSurrogate(int firstPart, int secondPart)
        {
            // Ok, then, is the second part valid?
            if (secondPart < SURR2_FIRST || secondPart > SURR2_LAST) {
                throw new IllegalArgumentException("Broken surrogate pair: first char 0x"+Integer.toHexString(firstPart)+",
second 0x"+Integer.toHexString(secondPart)+"; illegal combination");
            }
            return 0x10000 + ((firstPart - SURR1_FIRST) << 10) + (secondPart - SURR2_FIRST);
        }

        private void _throwIllegalSurrogate(int code)
        {
            if (code > 0x10FFFF) { // over max?
                throw new IllegalArgumentException("Illegal character point (0x"+Integer.toHexString(code)+") to output;
max is 0x10FFFF as per RFC 4627");
            }
            if (code >= SURR1_FIRST) {
                if (code <= SURR1_LAST) { // Unmatched first part (closing without second part?)
                    throw new IllegalArgumentException("Unmatched first part of surrogate pair
(0x"+Integer.toHexString(code)+")");
                }
                throw new IllegalArgumentException("Unmatched second part of surrogate pair
(0x"+Integer.toHexString(code)+")");
            }
            // should we ever get this?
            throw new IllegalArgumentException("Illegal character point (0x"+Integer.toHexString(code)+") to output");
        }

    }
```

```
package org.codehaus.jackson.io;

import java.io.*;


public final class UTF8Writer
  extends Writer
{
  final static int SURR1_FIRST = 0xD800;
  final static int SURR1_LAST = 0xDBFF;
  final static int SURR2_FIRST = 0xDC00;
  final static int SURR2_LAST = 0xDFFF;

  final protected IOContext _context;

  OutputStream _out;

  byte[] _outBuffer;

  final int _outBufferEnd;

  int _outPtr;

  /**
   * When outputting chars from BMP, surrogate pairs need to be coalesced.
   * To do this, both pairs must be known first; and since it is possible
   * pairs may be split, we need temporary storage for the first half
   */
  int _surrogate = 0;

  public UTF8Writer(IOContext ctxt, OutputStream out)
  {
     _context = ctxt;
     _out = out;

     _outBuffer = ctxt.allocWriteEncodingBuffer();
     /* Max. expansion for a single char (in unmodified UTF-8) is
      * 4 bytes (or 3 depending on how you view it -- 4 when recombining
      * surrogate pairs)
      */
     _outBufferEnd = _outBuffer.length - 4;
     _outPtr = 0;
  }

  @Override
  public Writer append(char c)
     throws IOException
  {
```

```
    write(c);
    return this;
}

@Override
public void close()
    throws IOException
{
    if (_out != null) {
        if (_outPtr > 0) {
            _out.write(_outBuffer, 0, _outPtr);
            _outPtr = 0;
        }
        OutputStream out = _out;
        _out = null;

        byte[] buf = _outBuffer;
        if (buf != null) {
            _outBuffer = null;
            _context.releaseWriteEncodingBuffer(buf);
        }

        out.close();

        /* Let's 'flush' orphan surrogate, no matter what; but only
         * after cleanly closing everything else.
         */
        int code = _surrogate;
        _surrogate = 0;
        if (code > 0) {
            throwIllegal(code);
        }
    }
}

@Override
public void flush()
    throws IOException
{
    if (_outPtr > 0) {
        _out.write(_outBuffer, 0, _outPtr);
        _outPtr = 0;
    }
    _out.flush();
}

@Override
public void write(char[] cbuf)
```

```
    throws IOException
{
    write(cbuf, 0, cbuf.length);
}

@Override
public void write(char[] cbuf, int off, int len)
    throws IOException
{
    if (len < 2) {
        if (len == 1) {
            write(cbuf[off]);
        }
        return;
    }

    // First: do we have a leftover surrogate to deal with?
    if (_surrogate > 0) {
        char second = cbuf[off++];
        --len;
        write(convertSurrogate(second));
        // will have at least one more char
    }

    int outPtr = _outPtr;
    byte[] outBuf = _outBuffer;
    int outBufLast = _outBufferEnd; // has 4 'spare' bytes

    // All right; can just loop it nice and easy now:
    len += off; // len will now be the end of input buffer

    output_loop:
    for (; off < len; ) {
        /* First, let's ensure we can output at least 4 bytes
         * (longest UTF-8 encoded codepoint):
         */
        if (outPtr >= outBufLast) {
            _out.write(outBuf, 0, outPtr);
            outPtr = 0;
        }

        int c = cbuf[off++];
        // And then see if we have an Ascii char:
        if (c < 0x80) { // If so, can do a tight inner loop:
            outBuf[outPtr++] = (byte)c;
            // Let's calc how many ascii chars we can copy at most:
            int maxInCount = (len - off);
            int maxOutCount = (outBufLast - outPtr);
```

```
    if (maxInCount > maxOutCount) {
        maxInCount = maxOutCount;
    }
    maxInCount += off;
    ascii_loop:
    while (true) {
        if (off >= maxInCount) { // done with max. ascii seq
            continue output_loop;
        }
        c = cbuf[off++];
        if (c >= 0x80) {
            break ascii_loop;
        }
        outBuf[outPtr++] = (byte) c;
    }
}

// Nope, multi-byte:
if (c < 0x800) { // 2-byte
    outBuf[outPtr++] = (byte) (0xc0 | (c >> 6));
    outBuf[outPtr++] = (byte) (0x80 | (c & 0x3f));
} else { // 3 or 4 bytes
    // Surrogates?
    if (c < SURR1_FIRST || c > SURR2_LAST) {
        outBuf[outPtr++] = (byte) (0xe0 | (c >> 12));
        outBuf[outPtr++] = (byte) (0x80 | ((c >> 6) & 0x3f));
        outBuf[outPtr++] = (byte) (0x80 | (c & 0x3f));
        continue;
    }
    // Yup, a surrogate:
    if (c > SURR1_LAST) { // must be from first range
        _outPtr = outPtr;
        throwIllegal(c);
    }
    _surrogate = c;
    // and if so, followed by another from next range
    if (off >= len) { // unless we hit the end?
        break;
    }
    c = convertSurrogate(cbuf[off++]);
    if (c > 0x10FFFF) { // illegal in JSON as well as in XML
        _outPtr = outPtr;
        throwIllegal(c);
    }
    outBuf[outPtr++] = (byte) (0xf0 | (c >> 18));
    outBuf[outPtr++] = (byte) (0x80 | ((c >> 12) & 0x3f));
    outBuf[outPtr++] = (byte) (0x80 | ((c >> 6) & 0x3f));
```

```
            outBuf[outPtr++] = (byte) (0x80 | (c & 0x3f));
        }
    }
    _outPtr = outPtr;
}

@Override
public void write(int c) throws IOException
{
    // First; do we have a left over surrogate?
    if (_surrogate > 0) {
        c = convertSurrogate(c);
        // If not, do we start with a surrogate?
    } else if (c >= SURR1_FIRST && c <= SURR2_LAST) {
        // Illegal to get second part without first:
        if (c > SURR1_LAST) {
            throwIllegal(c);
        }
        // First part just needs to be held for now
        _surrogate = c;
        return;
    }

    if (_outPtr >= _outBufferEnd) { // let's require enough room, first
        _out.write(_outBuffer, 0, _outPtr);
        _outPtr = 0;
    }

    if (c < 0x80) { // ascii
        _outBuffer[_outPtr++] = (byte) c;
    } else {
        int ptr = _outPtr;
        if (c < 0x800) { // 2-byte
            _outBuffer[ptr++] = (byte) (0xc0 | (c >> 6));
            _outBuffer[ptr++] = (byte) (0x80 | (c & 0x3f));
        } else if (c <= 0xFFFF) { // 3 bytes
            _outBuffer[ptr++] = (byte) (0xe0 | (c >> 12));
            _outBuffer[ptr++] = (byte) (0x80 | ((c >> 6) & 0x3f));
            _outBuffer[ptr++] = (byte) (0x80 | (c & 0x3f));
        } else { // 4 bytes
            if (c > 0x10FFFF) { // illegal
                throwIllegal(c);
            }
            _outBuffer[ptr++] = (byte) (0xf0 | (c >> 18));
            _outBuffer[ptr++] = (byte) (0x80 | ((c >> 12) & 0x3f));
            _outBuffer[ptr++] = (byte) (0x80 | ((c >> 6) & 0x3f));
            _outBuffer[ptr++] = (byte) (0x80 | (c & 0x3f));
        }
```

```
            _outPtr = ptr;
        }
    }

    @Override
    public void write(String str) throws IOException
    {
        write(str, 0, str.length());
    }

    @Override
    public void write(String str, int off, int len)  throws IOException
    {
        if (len < 2) {
            if (len == 1) {
                write(str.charAt(off));
            }
            return;
        }

        // First: do we have a leftover surrogate to deal with?
        if (_surrogate > 0) {
            char second = str.charAt(off++);
            --len;
            write(convertSurrogate(second));
            // will have at least one more char (case of 1 char was checked earlier on)
        }

        int outPtr = _outPtr;
        byte[] outBuf = _outBuffer;
        int outBufLast = _outBufferEnd; // has 4 'spare' bytes

        // All right; can just loop it nice and easy now:
        len += off; // len will now be the end of input buffer

        output_loop:
        for (; off < len; ) {
            /* First, let's ensure we can output at least 4 bytes
             * (longest UTF-8 encoded codepoint):
             */
            if (outPtr >= outBufLast) {
                _out.write(outBuf, 0, outPtr);
                outPtr = 0;
            }

            int c = str.charAt(off++);
            // And then see if we have an Ascii char:
            if (c < 0x80) { // If so, can do a tight inner loop:
```

```
    outBuf[outPtr++] = (byte)c;
    // Let's calc how many ascii chars we can copy at most:
    int maxInCount = (len - off);
    int maxOutCount = (outBufLast - outPtr);

    if (maxInCount > maxOutCount) {
       maxInCount = maxOutCount;
    }
    maxInCount += off;
    ascii_loop:
    while (true) {
       if (off >= maxInCount) { // done with max. ascii seq
          continue output_loop;
       }
       c = str.charAt(off++);
       if (c >= 0x80) {
          break ascii_loop;
       }
       outBuf[outPtr++] = (byte) c;
    }
}

// Nope, multi-byte:
if (c < 0x800) { // 2-byte
    outBuf[outPtr++] = (byte) (0xc0 | (c >> 6));
    outBuf[outPtr++] = (byte) (0x80 | (c & 0x3f));
} else { // 3 or 4 bytes
    // Surrogates?
    if (c < SURR1_FIRST || c > SURR2_LAST) {
       outBuf[outPtr++] = (byte) (0xe0 | (c >> 12));
       outBuf[outPtr++] = (byte) (0x80 | ((c >> 6) & 0x3f));
       outBuf[outPtr++] = (byte) (0x80 | (c & 0x3f));
       continue;
    }
    // Yup, a surrogate:
    if (c > SURR1_LAST) { // must be from first range
       _outPtr = outPtr;
       throwIllegal(c);
    }
    _surrogate = c;
    // and if so, followed by another from next range
    if (off >= len) { // unless we hit the end?
       break;
    }
    c = convertSurrogate(str.charAt(off++));
    if (c > 0x10FFFF) { // illegal, as per RFC 4627
       _outPtr = outPtr;
       throwIllegal(c);
```

```
            }
            outBuf[outPtr++] = (byte) (0xf0 | (c >> 18));
            outBuf[outPtr++] = (byte) (0x80 | ((c >> 12) & 0x3f));
            outBuf[outPtr++] = (byte) (0x80 | ((c >> 6) & 0x3f));
            outBuf[outPtr++] = (byte) (0x80 | (c & 0x3f));
          }
        }
        _outPtr = outPtr;
    }

    /*
    /**********************************************************
    /* Internal methods
    /**********************************************************
     */

    /**
     * Method called to calculate UTF codepoint, from a surrogate pair.
     */
    private int convertSurrogate(int secondPart)
        throws IOException
    {
        int firstPart = _surrogate;
        _surrogate = 0;

        // Ok, then, is the second part valid?
        if (secondPart < SURR2_FIRST || secondPart > SURR2_LAST) {
            throw new IOException("Broken surrogate pair: first char 0x"+Integer.toHexString(firstPart)+", second
0x"+Integer.toHexString(secondPart)+"; illegal combination");
        }
        return 0x10000 + ((firstPart - SURR1_FIRST) << 10) + (secondPart - SURR2_FIRST);
    }

    private void throwIllegal(int code)
        throws IOException
    {
        if (code > 0x10FFFF) { // over max?
            throw new IOException("Illegal character point (0x"+Integer.toHexString(code)+") to output; max is
0x10FFFF as per RFC 4627");
        }
        if (code >= SURR1_FIRST) {
            if (code <= SURR1_LAST) { // Unmatched first part (closing without second part?)
                throw new IOException("Unmatched first part of surrogate pair (0x"+Integer.toHexString(code)+")");
            }
            throw new IOException("Unmatched second part of surrogate pair (0x"+Integer.toHexString(code)+")");
        }

        // should we ever get this?
```

```
            throw new IOException("Illegal character point (0x"+Integer.toHexString(code)+") to output");
        }
    }
}
package org.codehaus.jackson.io;

import java.io.*;

/**
* Simple {@link InputStream} implementation that is used to "unwind" some
* data previously read from an input stream; so that as long as some of
* that data remains, it's returned; but as long as it's read, we'll
* just use data from the underlying original stream.
* This is similar to {@link java.io.PushbackInputStream}, but here there's
* only one implicit pushback, when instance is constructed.
*/
public final class MergedStream
    extends InputStream
{
    final protected IOContext _context;

    final InputStream _in;

    byte[] _buffer;

    int _ptr;

    final int _end;

    public MergedStream(IOContext context,
                        InputStream in, byte[] buf, int start, int end)
    {
        _context = context;
        _in = in;
        _buffer = buf;
        _ptr = start;
        _end = end;
    }

    @Override
    public int available() throws IOException
    {
        if (_buffer != null) {
            return _end - _ptr;
        }
        return _in.available();
    }

    @Override
```

```
public void close() throws IOException
{
   freeMergedBuffer();
   _in.close();
}

@Override
public void mark(int readlimit)
{
   if (_buffer == null) {
      _in.mark(readlimit);
   }
}

@Override
public boolean markSupported()
{
   // Only supports marks past the initial rewindable section...
   return (_buffer == null) && _in.markSupported();
}

@Override
public int read() throws IOException
{
   if (_buffer != null) {
      int c = _buffer[_ptr++] & 0xFF;
      if (_ptr >= _end) {
         freeMergedBuffer();
      }
      return c;
   }
   return _in.read();
}

@Override
public int read(byte[] b) throws IOException
{
   return read(b, 0, b.length);
}

@Override
public int  read(byte[] b, int off, int len) throws IOException
{
   if (_buffer != null) {
      int avail = _end - _ptr;
      if (len > avail) {
         len = avail;
      }
```

```
            System.arraycopy(_buffer, _ptr, b, off, len);
            _ptr += len;
            if (_ptr >= _end) {
               freeMergedBuffer();
            }
            return len;
         }
         return _in.read(b, off, len);
      }

      @Override
      public void reset() throws IOException
      {
         if (_buffer == null) {
            _in.reset();
         }
      }

      @Override
      public long skip(long n) throws IOException
      {
         long count = 0L;

         if (_buffer != null) {
            int amount = _end - _ptr;

            if (amount > n) { // all in pushed back segment?
               _ptr += (int) n;
               return n;
            }
            freeMergedBuffer();
            count += amount;
            n -= amount;
         }

         if (n > 0) {
            count += _in.skip(n);
         }
         return count;
      }

      private void freeMergedBuffer()
      {
         byte[] buf = _buffer;
         if (buf != null) {
            _buffer = null;
            _context.releaseReadIOBuffer(buf);
         }
```

```
  }
}
package org.codehaus.jackson.io;

import org.codehaus.jackson.JsonEncoding;
import org.codehaus.jackson.util.BufferRecycler;
import org.codehaus.jackson.util.TextBuffer;

/**
* To limit number of configuration and state objects to pass, all
* contextual objects that need to be passed by the factory to
* readers and writers are combined under this object. One instance
* is created for each reader and writer.
*/
public final class IOContext
{
  /*
  /**********************************************************
  /* Configuration
  /**********************************************************
   */

  /**
   * Reference to the source object, which can be used for displaying
   * location information
   */
  protected final Object _sourceRef;

  /**
   * Encoding used by the underlying stream, if known.
   */
  protected JsonEncoding _encoding;

  /**
   * Flag that indicates whether underlying input/output source/target
   * object is fully managed by the owner of this context (parser or
   * generator). If true, it is, and is to be closed by parser/generator;
   * if false, calling application has to do closing (unless auto-closing
   * feature is enabled for the parser/generator in question; in which
   * case it acts like the owner).
   */
  protected final boolean _managedResource;

  /*
  /**********************************************************
  /* Buffer handling, recycling
  /**********************************************************
   */
```

```
/**
 * Recycler used for actual allocation/deallocation/reuse
 */
protected final BufferRecycler _bufferRecycler;

/**
 * Reference to the allocated I/O buffer for low-level input reading,
 * if any allocated.
 */
protected byte[] _readIOBuffer = null;

/**
 * Reference to the allocated I/O buffer used for low-level
 * encoding-related buffering.
 */
protected byte[] _writeEncodingBuffer = null;

/**
 * Reference to the buffer allocated for tokenization purposes,
 * in which character input is read, and from which it can be
 * further returned.
 */
protected char[] _tokenCBuffer = null;

/**
 * Reference to the buffer allocated for buffering it for
 * output, before being encoded: generally this means concatenating
 * output, then encoding when buffer fills up.
 */
protected char[] _concatCBuffer = null;

/**
 * Reference temporary buffer Parser instances need if calling
 * app decides it wants to access name via 'getTextCharacters' method.
 * Regular text buffer can not be used as it may contain textual
 * representation of the value token.
 */
protected char[] _nameCopyBuffer = null;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public IOContext(BufferRecycler br, Object sourceRef, boolean managedResource)
{
```

```
    _bufferRecycler = br;
    _sourceRef = sourceRef;
    _managedResource = managedResource;
}

public void setEncoding(JsonEncoding enc)
{
    _encoding = enc;
}

/*
/**********************************************************
/* Public API, accessors
/**********************************************************
 */

public final Object getSourceReference() { return _sourceRef; }
public final JsonEncoding getEncoding() { return _encoding; }
public final boolean isResourceManaged() { return _managedResource; }

/*
/**********************************************************
/* Public API, buffer management
/**********************************************************
 */

public final TextBuffer constructTextBuffer() {
    return new TextBuffer(_bufferRecycler);
}

/**
 *<p>
 * Note: the method can only be called once during its life cycle.
 * This is to protect against accidental sharing.
 */
public final byte[] allocReadIOBuffer()
{
    if (_readIOBuffer != null) {
        throw new IllegalStateException("Trying to call allocReadIOBuffer() second time");
    }
    _readIOBuffer = _bufferRecycler.allocByteBuffer(BufferRecycler.ByteBufferType.READ_IO_BUFFER);
    return _readIOBuffer;
}

public final byte[] allocWriteEncodingBuffer()
{
    if (_writeEncodingBuffer != null) {
        throw new IllegalStateException("Trying to call allocWriteEncodingBuffer() second time");
```

```
        }
        _writeEncodingBuffer =
    _bufferRecycler.allocByteBuffer(BufferRecycler.ByteBufferType.WRITE_ENCODING_BUFFER);
        return _writeEncodingBuffer;
    }

    public final char[] allocTokenBuffer()
    {
        if (_tokenCBuffer != null) {
            throw new IllegalStateException("Trying to call allocTokenBuffer() second time");
        }
        _tokenCBuffer = _bufferRecycler.allocCharBuffer(BufferRecycler.CharBufferType.TOKEN_BUFFER);
        return _tokenCBuffer;
    }

    public final char[] allocConcatBuffer()
    {
        if (_concatCBuffer != null) {
            throw new IllegalStateException("Trying to call allocConcatBuffer() second time");
        }
        _concatCBuffer = _bufferRecycler.allocCharBuffer(BufferRecycler.CharBufferType.CONCAT_BUFFER);
        return _concatCBuffer;
    }

    public final char[] allocNameCopyBuffer(int minSize)
    {
        if (_nameCopyBuffer != null) {
            throw new IllegalStateException("Trying to call allocNameCopyBuffer() second time");
        }
        _nameCopyBuffer =
    _bufferRecycler.allocCharBuffer(BufferRecycler.CharBufferType.NAME_COPY_BUFFER, minSize);
        return _nameCopyBuffer;
    }

    /**
     * Method to call when all the processing buffers can be safely
     * recycled.
     */
    public final void releaseReadIOBuffer(byte[] buf)
    {
        if (buf != null) {
            /* Let's do sanity checks to ensure once-and-only-once release,
             * as well as avoiding trying to release buffers not owned
             */
            if (buf != _readIOBuffer) {
                throw new IllegalArgumentException("Trying to release buffer not owned by the context");
            }
            _readIOBuffer = null;
```

```
        _bufferRecycler.releaseByteBuffer(BufferRecycler.ByteBufferType.READ_IO_BUFFER, buf);
    }
}

public final void releaseWriteEncodingBuffer(byte[] buf)
{
    if (buf != null) {
        /* Let's do sanity checks to ensure once-and-only-once release,
         * as well as avoiding trying to release buffers not owned
         */
        if (buf != _writeEncodingBuffer) {
            throw new IllegalArgumentException("Trying to release buffer not owned by the context");
        }
        _writeEncodingBuffer = null;
        _bufferRecycler.releaseByteBuffer(BufferRecycler.ByteBufferType.WRITE_ENCODING_BUFFER, buf);
    }
}

public final void releaseTokenBuffer(char[] buf)
{
    if (buf != null) {
        if (buf != _tokenCBuffer) {
            throw new IllegalArgumentException("Trying to release buffer not owned by the context");
        }
        _tokenCBuffer = null;
        _bufferRecycler.releaseCharBuffer(BufferRecycler.CharBufferType.TOKEN_BUFFER, buf);
    }
}

public final void releaseConcatBuffer(char[] buf)
{
    if (buf != null) {
        if (buf != _concatCBuffer) {
            throw new IllegalArgumentException("Trying to release buffer not owned by the context");
        }
        _concatCBuffer = null;
        _bufferRecycler.releaseCharBuffer(BufferRecycler.CharBufferType.CONCAT_BUFFER, buf);
    }
}

public final void releaseNameCopyBuffer(char[] buf)
{
    if (buf != null) {
        if (buf != _nameCopyBuffer) {
            throw new IllegalArgumentException("Trying to release buffer not owned by the context");
        }
        _nameCopyBuffer = null;
        _bufferRecycler.releaseCharBuffer(BufferRecycler.CharBufferType.NAME_COPY_BUFFER, buf);
```

```
    }
  }
}
package org.codehaus.jackson.util;

import java.io.*;
import java.util.regex.Pattern;

import org.codehaus.jackson.Version;

/**
* Functionality for supporting exposing of component {@link Version}s.
*
* @since 1.6
*/
public class VersionUtil
{
  public final static String VERSION_FILE = "VERSION.txt";

  private final static Pattern VERSION_SEPARATOR = Pattern.compile("[-_./;:]");

  /**
   * Helper method that will try to load version information for specified
   * class. Implementation is simple: class loader that loaded specified
   * class is asked to load resource with name "VERSION" from same
   * location (package) as class itself had.
   * If no version information is found, {@link Version#unknownVersion()} is
   * returned.
   */
  public static Version versionFor(Class<?> cls)
  {
    InputStream in;
    Version version = null;

    try {
      in = cls.getResourceAsStream(VERSION_FILE);
      if (in != null) {
        try {
          BufferedReader br = new BufferedReader(new InputStreamReader(in, "UTF-8"));
          version = parseVersion(br.readLine());
        } finally {
          try {
            in.close();
          } catch (IOException e) {
            throw new RuntimeException(e);
          }
        }
      }
```

```
        } catch (IOException e) { }
        return (version == null) ? Version.unknownVersion() : version;
    }

    public static Version parseVersion(String versionStr)
    {
        if (versionStr == null) return null;
        versionStr = versionStr.trim();
        if (versionStr.length() == 0) return null;
        String[] parts = VERSION_SEPARATOR.split(versionStr);
        // Let's not bother if there's no separate parts; otherwise use whatever we got
        if (parts.length < 2) {
            return null;
        }
        int major = parseVersionPart(parts[0]);
        int minor = parseVersionPart(parts[1]);
        int patch = (parts.length > 2) ? parseVersionPart(parts[2]) : 0;
        String snapshot = (parts.length > 3) ? parts[3] : null;
        return new Version(major, minor, patch, snapshot);
    }

    protected static int parseVersionPart(String partStr)
    {
        partStr = partStr.toString();
        int len = partStr.length();
        int number = 0;
        for (int i = 0; i < len; ++i) {
            char c = partStr.charAt(i);
            if (c > '9' || c < '0') break;
            number = (number * 10) + (c - '0');
        }
        return number;
    }
}
package org.codehaus.jackson.util;

import java.io.IOException;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.PrettyPrinter;

/**
* {@link PrettyPrinter} implementation that adds no indentation,
* just implements everything necessary for value output to work
* as expected, and provide simpler extension points to allow
* for creating simple custom implementations that add specific
* decoration or overrides. Since behavior then is very similar
```

```
* to using no pretty printer at all, usually sub-classes are used.
*<p>
* Beyond purely minimal implementation, there is limited amount of
* configurability which may be useful for actual use: for example,
* it is possible to redefine separator used between root-level
* values (default is single space; can be changed to line-feed).
*
* @since 1.6
*/
public class MinimalPrettyPrinter
  implements PrettyPrinter
{
  /**
   * Default String used for separating root values is single space.
   */
  public final static String DEFAULT_ROOT_VALUE_SEPARATOR = " ";

  protected String _rootValueSeparator = DEFAULT_ROOT_VALUE_SEPARATOR;

  /*
  /**********************************************************
  /* Life-cycle, construction, configuration
  /**********************************************************
   */

  public MinimalPrettyPrinter() { }

  public void setRootValueSeparator(String sep) {
     _rootValueSeparator = sep;
  }

  /*
  /**********************************************************
  /* PrettyPrinter impl
  /**********************************************************
   */

  @Override
  public void writeRootValueSeparator(JsonGenerator jg) throws IOException, JsonGenerationException
  {
     if (_rootValueSeparator != null) {
        jg.writeRaw(_rootValueSeparator);
     }
  }

  public void writeStartObject(JsonGenerator jg)
     throws IOException, JsonGenerationException
  {
```

```
    jg.writeRaw('{');
}

public void beforeObjectEntries(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    // nothing special, since no indentation is added
}

/**
 * Method called after an object field has been output, but
 * before the value is output.
 *<p>
 * Default handling will just output a single
 * colon to separate the two, without additional spaces.
 */
public void writeObjectFieldValueSeparator(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    jg.writeRaw(':');
}

/**
 * Method called after an object entry (field:value) has been completely
 * output, and before another value is to be output.
 *<p>
 * Default handling (without pretty-printing) will output a single
 * comma to separate the two.
 */
public void writeObjectEntrySeparator(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    jg.writeRaw(',');
}

public void writeEndObject(JsonGenerator jg, int nrOfEntries)
    throws IOException, JsonGenerationException
{
    jg.writeRaw('}');
}

public void writeStartArray(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    jg.writeRaw('[');
}

public void beforeArrayValues(JsonGenerator jg)
```

```
        throws IOException, JsonGenerationException
    {
        // nothing special, since no indentation is added
    }

    /**
     * Method called after an array value has been completely
     * output, and before another value is to be output.
     *<p>
     * Default handling (without pretty-printing) will output a single
     * comma to separate values.
     */
    public void writeArrayValueSeparator(JsonGenerator jg)
        throws IOException, JsonGenerationException
    {
        jg.writeRaw(',');
    }

    public void writeEndArray(JsonGenerator jg, int nrOfValues)
        throws IOException, JsonGenerationException
    {
        jg.writeRaw(']');
    }
}
package org.codehaus.jackson.util;

import java.util.Map;
import java.util.LinkedHashMap;

/**
* Singleton class that adds a simple first-level cache in front of
* regular String.intern() functionality. This is done as a minor
* performance optimization, to avoid calling native intern() method
* in cases where same String is being interned multiple times.
*<p>
* Note: that this class extends {@link LinkedHashMap} is an implementation
* detail -- no code should ever directly call Map methods.
*/
@SuppressWarnings("serial")
public final class InternCache
    extends LinkedHashMap<String,String>
{
    /**
     * Size to use is somewhat arbitrary, so let's choose something that's
     * neither too small (low hit ratio) nor too large (waste of memory)
     */
    private final static int MAX_ENTRIES = 192;
```

```
  public final static InternCache instance = new InternCache();

  private InternCache() {
     super(MAX_ENTRIES, 0.8f, true);
  }

  @Override
  protected boolean removeEldestEntry(Map.Entry<String,String> eldest)
  {
     return size() > MAX_ENTRIES;
  }

  public synchronized String intern(String input)
  {
     String result = get(input);
     if (result == null) {
        result = input.intern();
        put(result, result);
     }
     return result;
  }


}
package org.codehaus.jackson.util;

/**
* This is a small utility class, whose main functionality is to allow
* simple reuse of raw byte/char buffers. It is usually used through
* <code>ThreadLocal</code> member of the owning class pointing to
* instance of this class through a <code>SoftReference</code>. The
* end result is a low-overhead GC-cleanable recycling: hopefully
* ideal for use by stream readers.
*/
public class BufferRecycler
{
  public final static int DEFAULT_WRITE_CONCAT_BUFFER_LEN = 2000;

  public enum ByteBufferType {
     READ_IO_BUFFER(4000)
     /**
      * Buffer used for temporarily storing encoded content; used
      * for example by UTF-8 encoding writer
      */
     ,WRITE_ENCODING_BUFFER(4000)

     /**
      * Buffer used for temporarily concatenating output; used for
```

```
     * example when requesting output as byte array.
     */
    ,WRITE_CONCAT_BUFFER(2000)
    ;

    private final int size;

    ByteBufferType(int size) { this.size = size; }
}

public enum CharBufferType {
    TOKEN_BUFFER(2000) // Tokenizable input
        ,CONCAT_BUFFER(2000) // concatenated output
        ,TEXT_BUFFER(200) // Text content from input
        ,NAME_COPY_BUFFER(200) // Temporary buffer for getting name characters
        ;

    private final int size;

    CharBufferType(int size) { this.size = size; }
}

final protected byte[][] _byteBuffers = new byte[ByteBufferType.values().length][];
final protected char[][] _charBuffers = new char[CharBufferType.values().length][];

public BufferRecycler() { }

public final byte[] allocByteBuffer(ByteBufferType type)
{
    int ix = type.ordinal();
    byte[] buffer = _byteBuffers[ix];
    if (buffer == null) {
        buffer = balloc(type.size);
    } else {
        _byteBuffers[ix] = null;
    }
    return buffer;
}

public final void releaseByteBuffer(ByteBufferType type, byte[] buffer)
{
    _byteBuffers[type.ordinal()] = buffer;
}

public final char[] allocCharBuffer(CharBufferType type)
{
    return allocCharBuffer(type, 0);
}
```

```
  public final char[] allocCharBuffer(CharBufferType type, int minSize)
  {
     if (type.size > minSize) {
        minSize = type.size;
     }
     int ix = type.ordinal();
     char[] buffer = _charBuffers[ix];
     if (buffer == null || buffer.length < minSize) {
        buffer = calloc(minSize);
     } else {
        _charBuffers[ix] = null;
     }
     return buffer;
  }

  public final void releaseCharBuffer(CharBufferType type, char[] buffer)
  {
     _charBuffers[type.ordinal()] = buffer;
  }

  /*
  /**********************************************************
  /* Actual allocations separated for easier debugging/profiling
  /**********************************************************
   */

  private final byte[] balloc(int size)
  {
     return new byte[size];
  }

  private final char[] calloc(int size)
  {
     return new char[size];
  }
}
package org.codehaus.jackson.util;

import java.math.BigDecimal;
import java.util.ArrayList;

import org.codehaus.jackson.io.NumberInput;

/**
* TextBuffer is a class similar to {@link StringBuffer}, with
* following differences:
*<ul>
```

```
*  <li>TextBuffer uses segments character arrays, to avoid having
*     to do additional array copies when array is not big enough.
*     This means that only reallocating that is necessary is done only once:
*     if and when caller
*     wants to access contents in a linear array (char[], String).
*    </li>
*  <li>TextBuffer can also be initialized in "shared mode", in which
*     it will just act as a wrapper to a single char array managed
*     by another object (like parser that owns it)
*    </li>
*  <li>TextBuffer is not synchronized.
*    </li>
* </ul>
*/
public final class TextBuffer
{
  final static char[] NO_CHARS = new char[0];

  /**
   * Let's start with sizable but not huge buffer, will grow as necessary
   */
  final static int MIN_SEGMENT_LEN = 1000;

  /**
   * Let's limit maximum segment length to something sensible
   * like 256k
   */
  final static int MAX_SEGMENT_LEN = 0x40000;

  /*
  /**********************************************************
  /* Configuration:
  /**********************************************************
   */

  private final BufferRecycler _allocator;

  /*
  /**********************************************************
  /* Shared input buffers
  /**********************************************************
   */

  /**
   * Shared input buffer; stored here in case some input can be returned
   * as is, without being copied to collector's own buffers. Note that
   * this is read-only for this Object.
   */
```

```
private char[] _inputBuffer;

/**
 * Character offset of first char in input buffer; -1 to indicate
 * that input buffer currently does not contain any useful char data
 */
private int _inputStart;

private int _inputLen;

/*
/**********************************************************
/* Aggregation segments (when not using input buf)
/**********************************************************
 */

/**
 * List of segments prior to currently active segment.
 */
private ArrayList<char[]> _segments;

/**
 * Flag that indicates whether _seqments is non-empty
 */
private boolean _hasSegments = false;

// // // Currently used segment; not (yet) contained in _seqments

/**
 * Amount of characters in segments in {@link _segments}
 */
private int _segmentSize;

private char[] _currentSegment;

/**
 * Number of characters in currently active (last) segment
 */
private int _currentSize;

/*
/**********************************************************
/* Caching of results
/**********************************************************
 */

/**
 * String that will be constructed when the whole contents are
```

```
     * needed; will be temporarily stored in case asked for again.
     */
    private String _resultString;

    private char[] _resultArray;

    /*
    /**********************************************************
    /* Life-cycle
    /**********************************************************
     */

    public TextBuffer(BufferRecycler allocator)
    {
        _allocator = allocator;
    }

    /**
     * Method called to indicate that the underlying buffers should now
     * be recycled if they haven't yet been recycled. Although caller
     * can still use this text buffer, it is not advisable to call this
     * method if that is likely, since next time a buffer is needed,
     * buffers need to reallocated.
     * Note: calling this method automatically also clears contents
     * of the buffer.
     */
    public void releaseBuffers()
    {
        if (_allocator == null) {
            resetWithEmpty();
        } else {
            if (_currentSegment != null) {
                // First, let's get rid of all but the largest char array
                resetWithEmpty();
                // And then return that array
                char[] buf = _currentSegment;
                _currentSegment = null;
                _allocator.releaseCharBuffer(BufferRecycler.CharBufferType.TEXT_BUFFER, buf);
            }
        }
    }

    /**
     * Method called to clear out any content text buffer may have, and
     * initializes buffer to use non-shared data.
     */
    public void resetWithEmpty()
    {
```

```
    _inputStart = -1; // indicates shared buffer not used
    _currentSize = 0;
    _inputLen = 0;

    _inputBuffer = null;
    _resultString = null;
    _resultArray = null;

    // And then reset internal input buffers, if necessary:
    if (_hasSegments) {
        clearSegments();
    }
}

/**
 * Method called to initialize the buffer with a shared copy of data;
 * this means that buffer will just have pointers to actual data. It
 * also means that if anything is to be appended to the buffer, it
 * will first have to unshare it (make a local copy).
 */
public void resetWithShared(char[] buf, int start, int len)
{
    // First, let's clear intermediate values, if any:
    _resultString = null;
    _resultArray = null;

    // Then let's mark things we need about input buffer
    _inputBuffer = buf;
    _inputStart = start;
    _inputLen = len;

    // And then reset internal input buffers, if necessary:
    if (_hasSegments) {
        clearSegments();
    }
}

public void resetWithCopy(char[] buf, int start, int len)
{
    _inputBuffer = null;
    _inputStart = -1; // indicates shared buffer not used
    _inputLen = 0;

    _resultString = null;
    _resultArray = null;

    // And then reset internal input buffers, if necessary:
    if (_hasSegments) {
```

```
            clearSegments();
        } else if (_currentSegment == null) {
            _currentSegment = findBuffer(len);
        }
        _currentSize = _segmentSize = 0;
        append(buf, start, len);
    }

    public void resetWithString(String value)
    {
        _inputBuffer = null;
        _inputStart = -1;
        _inputLen = 0;

        _resultString = value;
        _resultArray = null;

        if (_hasSegments) {
            clearSegments();
        }
        _currentSize = 0;

    }

    /**
     * Helper method used to find a buffer to use, ideally one
     * recycled earlier.
     */
    private final char[] findBuffer(int needed)
    {
        if (_allocator != null) {
            return _allocator.allocCharBuffer(BufferRecycler.CharBufferType.TEXT_BUFFER, needed);
        }
        return new char[Math.max(needed, MIN_SEGMENT_LEN)];
    }

    private final void clearSegments()
    {
        _hasSegments = false;
        /* Let's start using _last_ segment from list; for one, it's
         * the biggest one, and it's also most likely to be cached
         */
        /* 28-Aug-2009, tatu: Actually, the current segment should
         *   be the biggest one, already
         */
        //_currentSegment = _segments.get(_segments.size() - 1);
        _segments.clear();
        _currentSize = _segmentSize = 0;
```

```
}

/*
/**********************************************************
/* Accessors for implementing public interface
/**********************************************************
 */

/**
 * @return Number of characters currently stored by this collector
 */
public int size() {
    if (_inputStart >= 0) { // shared copy from input buf
        return _inputLen;
    }
    // local segmented buffers
    return _segmentSize + _currentSize;
}

public int getTextOffset()
{
    /* Only shared input buffer can have non-zero offset; buffer
     * segments start at 0, and if we have to create a combo buffer,
     * that too will start from beginning of the buffer
     */
    return (_inputStart >= 0) ? _inputStart : 0;
}

public char[] getTextBuffer()
{
    // Are we just using shared input buffer?
    if (_inputStart >= 0) {
        return _inputBuffer;
    }
    // Nope; but does it fit in just one segment?
    if (!_hasSegments) {
        return _currentSegment;
    }
    // Nope, need to have/create a non-segmented array and return it
    return contentsAsArray();
}

/*
/**********************************************************
/* Other accessors:
/**********************************************************
 */
```

```
public String contentsAsString()
{
    if (_resultString == null) {
        // Has array been requested? Can make a shortcut, if so:
        if (_resultArray != null) {
            _resultString = new String(_resultArray);
        } else {
            // Do we use shared array?
            if (_inputStart >= 0) {
                if (_inputLen < 1) {
                    return (_resultString = "");
                }
                _resultString = new String(_inputBuffer, _inputStart, _inputLen);
            } else { // nope... need to copy
                // But first, let's see if we have just one buffer
                int segLen = _segmentSize;
                int currLen = _currentSize;

                if (segLen == 0) { // yup
                    _resultString = (currLen == 0) ? "" : new String(_currentSegment, 0, currLen);
                } else { // no, need to combine
                    StringBuilder sb = new StringBuilder(segLen + currLen);
                    // First stored segments
                    if (_segments != null) {
                        for (int i = 0, len = _segments.size(); i < len; ++i) {
                            char[] curr = _segments.get(i);
                            sb.append(curr, 0, curr.length);
                        }
                    }
                    // And finally, current segment:
                    sb.append(_currentSegment, 0, _currentSize);
                    _resultString = sb.toString();
                }
            }
        }
    }
    return _resultString;
}

public char[] contentsAsArray()
{
    char[] result = _resultArray;
    if (result == null) {
        _resultArray = result = buildResultArray();
    }
    return result;
}
```

```
/**
 * Convenience method for converting contents of the buffer
 * into a {@link BigDecimal}.
 */
public BigDecimal contentsAsDecimal()
   throws NumberFormatException
{
   // Already got a pre-cut array?
   if (_resultArray != null) {
      return new BigDecimal(_resultArray);
   }
   // Or a shared buffer?
   if (_inputStart >= 0) {
      return new BigDecimal(_inputBuffer, _inputStart, _inputLen);
   }
   // Or if not, just a single buffer (the usual case)
   if (_segmentSize == 0) {
      return new BigDecimal(_currentSegment, 0, _currentSize);
   }
   // If not, let's just get it aggregated...
   return new BigDecimal(contentsAsArray());
}

/**
 * Convenience method for converting contents of the buffer
 * into a Double value.
 */
public double contentsAsDouble()
   throws NumberFormatException
{
   return NumberInput.parseDouble(contentsAsString());
}

/*
/**********************************************************
/* Public mutators:
/**********************************************************
 */

/**
 * Method called to make sure that buffer is not using shared input
 * buffer; if it is, it will copy such contents to private buffer.
 */
public void ensureNotShared() {
   if (_inputStart >= 0) {
      unshare(16);
   }
}
```

```
public void append(char c) {
    // Using shared buffer so far?
    if (_inputStart >= 0) {
        unshare(16);
    }
    _resultString = null;
    _resultArray = null;
    // Room in current segment?
    char[] curr = _currentSegment;
    if (_currentSize >= curr.length) {
        expand(1);
        curr = _currentSegment;
    }
    curr[_currentSize++] = c;
}

public void append(char[] c, int start, int len)
{
    // Can't append to shared buf (sanity check)
    if (_inputStart >= 0) {
        unshare(len);
    }
    _resultString = null;
    _resultArray = null;

    // Room in current segment?
    char[] curr = _currentSegment;
    int max = curr.length - _currentSize;

    if (max >= len) {
        System.arraycopy(c, start, curr, _currentSize, len);
        _currentSize += len;
    } else {
        // No room for all, need to copy part(s):
        if (max > 0) {
            System.arraycopy(c, start, curr, _currentSize, max);
            start += max;
            len -= max;
        }
        // And then allocate new segment; we are guaranteed to now
        // have enough room in segment.
        expand(len); // note: curr != _currentSegment after this
        System.arraycopy(c, start, _currentSegment, 0, len);
        _currentSize = len;
    }
}
```

```
public void append(String str, int offset, int len)
{
    // Can't append to shared buf (sanity check)
    if (_inputStart >= 0) {
        unshare(len);
    }
    _resultString = null;
    _resultArray = null;

    // Room in current segment?
    char[] curr = _currentSegment;
    int max = curr.length - _currentSize;
    if (max >= len) {
        str.getChars(offset, offset+len, curr, _currentSize);
        _currentSize += len;
    } else {
        // No room for all, need to copy part(s):
        if (max > 0) {
            str.getChars(offset, offset+max, curr, _currentSize);
            len -= max;
            offset += max;
        }
        /* And then allocate new segment; we are guaranteed to now
         * have enough room in segment.
         */
        expand(len);
        str.getChars(offset, offset+len, _currentSegment, 0);
        _currentSize = len;
    }
}

/*
/**********************************************************
/* Raw access, for high-performance use:
/**********************************************************
 */

public char[] getCurrentSegment()
{
    /* Since the intention of the caller is to directly add stuff into
     * buffers, we should NOT have anything in shared buffer... ie. may
     * need to unshare contents.
     */
    if (_inputStart >= 0) {
        unshare(1);
    } else {
        char[] curr = _currentSegment;
        if (curr == null) {
```

```
            _currentSegment = findBuffer(0);
        } else if (_currentSize >= curr.length) {
            // Plus, we better have room for at least one more char
            expand(1);
        }
    }
    return _currentSegment;
}

public final char[] emptyAndGetCurrentSegment()
{
    // inlined 'resetWithEmpty()'
    _inputStart = -1; // indicates shared buffer not used
    _currentSize = 0;
    _inputLen = 0;

    _inputBuffer = null;
    _resultString = null;
    _resultArray = null;

    // And then reset internal input buffers, if necessary:
    if (_hasSegments) {
        clearSegments();
    }
    char[] curr = _currentSegment;
    if (curr == null) {
        _currentSegment = curr = findBuffer(0);
    }
    return curr;
}

public int getCurrentSegmentSize() {
    return _currentSize;
}

public void setCurrentLength(int len) {
    _currentSize = len;
}

public char[] finishCurrentSegment()
{
    if (_segments == null) {
        _segments = new ArrayList<char[]>();
    }
    _hasSegments = true;
    _segments.add(_currentSegment);
    int oldLen = _currentSegment.length;
    _segmentSize += oldLen;
```

```
    // Let's grow segments by 50%
    int newLen = Math.min(oldLen + (oldLen >> 1), MAX_SEGMENT_LEN);
    char[] curr = _charArray(newLen);
    _currentSize = 0;
    _currentSegment = curr;
    return curr;
}

/**
 * Method called to expand size of the current segment, to
 * accomodate for more contiguous content. Usually only
 * used when parsing tokens like names.
 */
public char[] expandCurrentSegment()
{
    char[] curr = _currentSegment;
    // Let's grow by 50%
    int len = curr.length;
    // Must grow by at least 1 char, no matter what
    int newLen = (len == MAX_SEGMENT_LEN) ?
      (MAX_SEGMENT_LEN + 1) : Math.min(MAX_SEGMENT_LEN, len + (len >> 1));
    _currentSegment = _charArray(newLen);
    System.arraycopy(curr, 0, _currentSegment, 0, len);
    return _currentSegment;
}

/*
/**********************************************************
/* Standard methods:
/**********************************************************
 */

/**
 * Note: calling this method may not be as efficient as calling
 * {@link #contentsAsString}, since it's not guaranteed that resulting
 * String is cached.
 */
@Override
public String toString() {
    return contentsAsString();
}

/*
/**********************************************************
/* Internal methods:
/**********************************************************
 */
```

```
/**
 * Method called if/when we need to append content when we have been
 * initialized to use shared buffer.
 */
private void unshare(int needExtra)
{
    int sharedLen = _inputLen;
    _inputLen = 0;
    char[] inputBuf = _inputBuffer;
    _inputBuffer = null;
    int start = _inputStart;
    _inputStart = -1;

    // Is buffer big enough, or do we need to reallocate?
    int needed = sharedLen+needExtra;
    if (_currentSegment == null || needed > _currentSegment.length) {
        _currentSegment = findBuffer(needed);
    }
    if (sharedLen > 0) {
        System.arraycopy(inputBuf, start, _currentSegment, 0, sharedLen);
    }
    _segmentSize = 0;
    _currentSize = sharedLen;
}

/**
 * Method called when current segment is full, to allocate new
 * segment.
 */
private void expand(int minNewSegmentSize)
{
    // First, let's move current segment to segment list:
    if (_segments == null) {
        _segments = new ArrayList<char[]>();
    }
    char[] curr = _currentSegment;
    _hasSegments = true;
    _segments.add(curr);
    _segmentSize += curr.length;
    int oldLen = curr.length;
    // Let's grow segments by 50% minimum
    int sizeAddition = oldLen >> 1;
    if (sizeAddition < minNewSegmentSize) {
        sizeAddition = minNewSegmentSize;
    }
    curr = _charArray(Math.min(MAX_SEGMENT_LEN, oldLen + sizeAddition));
    _currentSize = 0;
    _currentSegment = curr;
```

```
    }

    private char[] buildResultArray()
    {
        if (_resultString != null) { // Can take a shortcut...
            return _resultString.toCharArray();
        }
        char[] result;

        // Do we use shared array?
        if (_inputStart >= 0) {
            if (_inputLen < 1) {
                return NO_CHARS;
            }
            result = _charArray(_inputLen);
            System.arraycopy(_inputBuffer, _inputStart, result, 0,
                             _inputLen);
        } else { // nope
            int size = size();
            if (size < 1) {
                return NO_CHARS;
            }
            int offset = 0;
            result = _charArray(size);
            if (_segments != null) {
                for (int i = 0, len = _segments.size(); i < len; ++i) {
                    char[] curr = (char[]) _segments.get(i);
                    int currLen = curr.length;
                    System.arraycopy(curr, 0, result, offset, currLen);
                    offset += currLen;
                }
            }
            System.arraycopy(_currentSegment, 0, result, offset, _currentSize);
        }
        return result;
    }

    private final char[] _charArray(int len) {
        return new char[len];
    }
}
/* Jackson JSON-processor.
*
* Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi
*
* Licensed under the License specified in file LICENSE, included with
* the source code and binary code bundles.
* You may not use this file except in compliance with the License.
```

```
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package org.codehaus.jackson.util;

import java.io.OutputStream;
import java.util.*;

/**
* Helper class that is similar to {@link java.io.ByteArrayOutputStream}
* in usage, but more geared to Jackson use cases internally.
* Specific changes include segment storage (no need to have linear
* backing buffer, can avoid reallocs, copying), as well API
* not based on {@link java.io.OutputStream}. In short, a very much
* specialized builder object.
*<p>
* Since version 1.5, also implements {@link OutputStream} to allow
* efficient aggregation of output content as a byte array, similar
* to how {@link java.io.ByteArrayOutputStream} works, but somewhat more
* efficiently for many use cases.
*/
public final class ByteArrayBuilder
  extends OutputStream
{
  private final static byte[] NO_BYTES = new byte[0];

  /**
   * Size of the first block we will allocate.
   */
  private final static int INITIAL_BLOCK_SIZE = 500;

  /**
   * Maximum block size we will use for individual non-aggregated
   * blocks. Let's limit to using 256k chunks.
   */
  private final static int MAX_BLOCK_SIZE = (1 << 18);

  final static int DEFAULT_BLOCK_ARRAY_SIZE = 40;

  /**
   * Optional buffer recycler instance that we can use for allocating
   * the first block.
   *
```

```
 * @since 1.5
 */
private final BufferRecycler _bufferRecycler;

private final LinkedList<byte[]> _pastBlocks = new LinkedList<byte[]>();

/**
 * Number of bytes within byte arrays in {@link _pastBlocks}.
 */
private int _pastLen;

private byte[] _currBlock;

private int _currBlockPtr;

public ByteArrayBuilder() { this(null); }

public ByteArrayBuilder(BufferRecycler br) { this(br, INITIAL_BLOCK_SIZE); }

public ByteArrayBuilder(int firstBlockSize) { this(null, firstBlockSize); }

public ByteArrayBuilder(BufferRecycler br, int firstBlockSize)
{
   _bufferRecycler = br;
   if (br == null) {
      _currBlock = new byte[firstBlockSize];
   } else {
      _currBlock = br.allocByteBuffer(BufferRecycler.ByteBufferType.WRITE_CONCAT_BUFFER);
   }
}

public void reset()
{
   _pastLen = 0;
   _currBlockPtr = 0;

   if (!_pastBlocks.isEmpty()) {
      _pastBlocks.clear();
   }
}

/**
 * Clean up method to call to release all buffers this object may be
 * using. After calling the method, no other accessors can be used (and
 * attempt to do so may result in an exception)
 */
public void release() {
   reset();
```

```
        if (_bufferRecycler != null && _currBlock != null) {
            _bufferRecycler.releaseByteBuffer(BufferRecycler.ByteBufferType.WRITE_CONCAT_BUFFER,
_currBlock);
            _currBlock = null;
        }
    }

    public void append(int i)
    {
        if (_currBlockPtr >= _currBlock.length) {
            _allocMore();
        }
        _currBlock[_currBlockPtr++] = (byte) i;
    }

    public void appendTwoBytes(int b16)
    {
        if ((_currBlockPtr + 1) < _currBlock.length) {
            _currBlock[_currBlockPtr++] = (byte) (b16 >> 8);
            _currBlock[_currBlockPtr++] = (byte) b16;
        } else {
            append(b16 >> 8);
            append(b16);
        }
    }

    public void appendThreeBytes(int b24)
    {
        if ((_currBlockPtr + 2) < _currBlock.length) {
            _currBlock[_currBlockPtr++] = (byte) (b24 >> 16);
            _currBlock[_currBlockPtr++] = (byte) (b24 >> 8);
            _currBlock[_currBlockPtr++] = (byte) b24;
        } else {
            append(b24 >> 16);
            append(b24 >> 8);
            append(b24);
        }
    }

    /**
     * Method called when results are finalized and we can get the
     * full aggregated result buffer to return to the caller
     */
    public byte[] toByteArray()
    {
        int totalLen = _pastLen + _currBlockPtr;

        if (totalLen == 0) { // quick check: nothing aggregated?
```

```
            return NO_BYTES;
        }

        byte[] result = new byte[totalLen];
        int offset = 0;

        for (byte[] block : _pastBlocks) {
            int len = block.length;
            System.arraycopy(block, 0, result, offset, len);
            offset += len;
        }
        System.arraycopy(_currBlock, 0, result, offset, _currBlockPtr);
        offset += _currBlockPtr;
        if (offset != totalLen) { // just a sanity check
            throw new RuntimeException("Internal error: total len assumed to be "+totalLen+", copied "+offset+"
bytes");
        }
        // Let's only reset if there's sizable use, otherwise will get reset later on
        if (!_pastBlocks.isEmpty()) {
            reset();
        }
        return result;
    }

    /*
    /**********************************************************
    /* Non-stream API (similar to TextBuffer), since 1.6
    /**********************************************************
     */

    /**
     * Method called when starting "manual" output: will clear out
     * current state and return the first segment buffer to fill
     *
     * @since 1.6
     */
    public byte[] resetAndGetFirstSegment() {
        reset();
        return _currBlock;
    }

    /**
     * Method called when the current segment buffer is full; will
     * append to current contents, allocate a new segment buffer
     * and return it
     *
     * @since 1.6
     */
```

```
public byte[] finishCurrentSegment() {
   _allocMore();
   return _currBlock;
}

/**
 * Method that will complete "manual" output process, coalesce
 * content (if necessary) and return results as a contiguous buffer.
 *
 * @param lastBlockLength Amount of content in the current segment
 * buffer.
 *
 * @return Coalesced contents
 */
public byte[] completeAndCoalesce(int lastBlockLength)
{
   _currBlockPtr = lastBlockLength;
   return toByteArray();
}

public byte[] getCurrentSegment() {
   return _currBlock;
}

public void setCurrentSegmentLength(int len) {
   _currBlockPtr = len;
}

public int getCurrentSegmentLength() {
   return _currBlockPtr;
}

/*
/**********************************************************
/* OutputStream implementation
/**********************************************************
 */

@Override
public void write(byte[] b) {
   write(b, 0, b.length);
}

@Override
public void write(byte[] b, int off, int len)
{
   while (true) {
      int max = _currBlock.length - _currBlockPtr;
```

```
            int toCopy = Math.min(max, len);
            if (toCopy > 0) {
                System.arraycopy(b, off, _currBlock, _currBlockPtr, toCopy);
                off += toCopy;
                _currBlockPtr += toCopy;
                len -= toCopy;
            }
            if (len <= 0) break;
            _allocMore();
        }
    }

    @Override
    public void write(int b) {
        append(b);
    }

    @Override public void close() { /* NOP */ }

    @Override public void flush() { /* NOP */ }

    /*
    /**********************************************************
    /* Internal methods
    /**********************************************************
     */

    private void _allocMore()
    {
        _pastLen += _currBlock.length;

        /* Let's allocate block that's half the total size, except
         * never smaller than twice the initial block size.
         * The idea is just to grow with reasonable rate, to optimize
         * between minimal number of chunks and minimal amount of
         * wasted space.
         */
        int newSize = Math.max((_pastLen >> 1), (INITIAL_BLOCK_SIZE + INITIAL_BLOCK_SIZE));
        // plus not to exceed max we define...
        if (newSize > MAX_BLOCK_SIZE) {
            newSize = MAX_BLOCK_SIZE;
        }
        _pastBlocks.add(_currBlock);
        _currBlock = new byte[newSize];
        _currBlockPtr = 0;
    }

}
```

```
package org.codehaus.jackson.util;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.SerializedString;

public class JsonGeneratorDelegate extends JsonGenerator
{
  /**
   * Delegate object that method calls are delegated to.
   */
  protected JsonGenerator delegate;

  public JsonGeneratorDelegate(JsonGenerator d) {
     delegate = d;
  }

  @Override
  public void close() throws IOException {
     delegate.close();
  }

  @Override
  public void copyCurrentEvent(JsonParser jp) throws IOException, JsonProcessingException {
     delegate.copyCurrentEvent(jp);
  }

  @Override
  public void copyCurrentStructure(JsonParser jp) throws IOException, JsonProcessingException {
     delegate.copyCurrentStructure(jp);
  }

  @Override
  public JsonGenerator disable(Feature f) {
     return delegate.disable(f);
  }

  @Override
  public JsonGenerator enable(Feature f) {
     return delegate.enable(f);
  }

  @Override
  public void flush() throws IOException {
     delegate.flush();
```

```
}

@Override
public ObjectCodec getCodec() {
   return delegate.getCodec();
}

@Override
public JsonStreamContext getOutputContext() {
   return delegate.getOutputContext();
}

@Override
public boolean isClosed() {
   return delegate.isClosed();
}

@Override
public boolean isEnabled(Feature f) {
   return delegate.isEnabled(f);
}

@Override
public JsonGenerator setCodec(ObjectCodec oc) {
   delegate.setCodec(oc);
   return this;
}

@Override
public JsonGenerator useDefaultPrettyPrinter() {
   delegate.useDefaultPrettyPrinter();
   return this;
}

@Override
public void writeBinary(Base64Variant b64variant, byte[] data, int offset, int len)
   throws IOException, JsonGenerationException
{
   delegate.writeBinary(b64variant, data, offset, len);
}

@Override
public void writeBoolean(boolean state) throws IOException, JsonGenerationException {
   delegate.writeBoolean(state);
}

@Override
public void writeEndArray() throws IOException, JsonGenerationException {
```

```
    delegate.writeEndArray();
}

@Override
public void writeEndObject() throws IOException, JsonGenerationException {
    delegate.writeEndObject();
}

@Override
public void writeFieldName(String name)
    throws IOException, JsonGenerationException
{
    delegate.writeFieldName(name);
}

@Override
public void writeFieldName(SerializedString name)
    throws IOException, JsonGenerationException
{
    delegate.writeFieldName(name);
}

@Override
public void writeFieldName(SerializableString name)
    throws IOException, JsonGenerationException
{
    delegate.writeFieldName(name);
}

@Override
public void writeNull() throws IOException, JsonGenerationException {
    delegate.writeNull();
}

@Override
public void writeNumber(int v) throws IOException, JsonGenerationException {
    delegate.writeNumber(v);
}

@Override
public void writeNumber(long v) throws IOException, JsonGenerationException {
    delegate.writeNumber(v);
}

@Override
public void writeNumber(BigInteger v) throws IOException,
        JsonGenerationException {
    delegate.writeNumber(v);
```

```
    }

    @Override
    public void writeNumber(double v) throws IOException,
            JsonGenerationException {
        delegate.writeNumber(v);
    }

    @Override
    public void writeNumber(float v) throws IOException,
            JsonGenerationException {
        delegate.writeNumber(v);
    }

    @Override
    public void writeNumber(BigDecimal v) throws IOException,
            JsonGenerationException {
        delegate.writeNumber(v);
    }

    @Override
    public void writeNumber(String encodedValue) throws IOException, JsonGenerationException,
UnsupportedOperationException {
        delegate.writeNumber(encodedValue);
    }

    @Override
    public void writeObject(Object pojo) throws IOException,JsonProcessingException {
        delegate.writeObject(pojo);
    }

    @Override
    public void writeRaw(String text) throws IOException, JsonGenerationException {
        delegate.writeRaw(text);
    }

    @Override
    public void writeRaw(String text, int offset, int len) throws IOException, JsonGenerationException {
        delegate.writeRaw(text, offset, len);
    }

    @Override
    public void writeRaw(char[] text, int offset, int len) throws IOException, JsonGenerationException {
        delegate.writeRaw(text, offset, len);
    }

    @Override
    public void writeRaw(char c) throws IOException, JsonGenerationException {
```

```
    delegate.writeRaw(c);
}

@Override
public void writeRawValue(String text) throws IOException, JsonGenerationException {
    delegate.writeRawValue(text);
}

@Override
public void writeRawValue(String text, int offset, int len) throws IOException, JsonGenerationException {
     delegate.writeRawValue(text, offset, len);
}

@Override
public void writeRawValue(char[] text, int offset, int len) throws IOException, JsonGenerationException {
     delegate.writeRawValue(text, offset, len);
}

@Override
public void writeStartArray() throws IOException, JsonGenerationException {
     delegate.writeStartArray();
}

@Override
public void writeStartObject() throws IOException, JsonGenerationException {
    delegate.writeStartObject();
}

@Override
public void writeString(String text) throws IOException,JsonGenerationException {
    delegate.writeString(text);
}

@Override
public void writeString(char[] text, int offset, int len) throws IOException, JsonGenerationException {
    delegate.writeString(text, offset, len);
}

@Override
public void writeString(SerializableString text) throws IOException, JsonGenerationException {
    delegate.writeString(text);
}

@Override
public void writeRawUTF8String(byte[] text, int offset, int length)
    throws IOException, JsonGenerationException
{
    delegate.writeRawUTF8String(text, offset, length);
```

```
    }

    @Override
    public void writeUTF8String(byte[] text, int offset, int length)
        throws IOException, JsonGenerationException
    {
        delegate.writeUTF8String(text, offset, length);
    }

    @Override
    public void writeTree(JsonNode rootNode) throws IOException, JsonProcessingException {
        delegate.writeTree(rootNode);
    }
}
package org.codehaus.jackson.util;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;
import org.codehaus.jackson.impl.JsonParserMinimalBase;
import org.codehaus.jackson.impl.JsonReadContext;
import org.codehaus.jackson.impl.JsonWriteContext;
import org.codehaus.jackson.io.SerializedString;

/**
* Utility class used for efficient storage of {@link JsonToken}
* sequences, needed for temporary buffering.
* Space efficient for different sequence lengths (especially so for smaller
* ones; but not significantly less efficient for larger), highly efficient
* for linear iteration and appending. Implemented as segmented/chunked
* linked list of tokens; only modifications are via appends.
*
* @since 1.5
*/
public class TokenBuffer
/* Won't use JsonGeneratorBase, to minimize overhead for validity
* checking
*/
    extends JsonGenerator
{
    protected final static int DEFAULT_PARSER_FEATURES = JsonParser.Feature.collectDefaults();

    /*
    /**********************************************************
    /* Configuration
    /**********************************************************
```

```
 */

/**
 * Object codec to use for stream-based object
 *   conversion through parser/generator interfaces. If null,
 *   such methods can not be used.
 */
protected ObjectCodec _objectCodec;

/**
 * Bit flag composed of bits that indicate which
 * {@link org.codehaus.jackson.JsonGenerator.Feature}s
 * are enabled.
 *<p>
 * NOTE: most features have no effect on this class
 */
protected int _generatorFeatures;

protected boolean _closed;

/*
/**********************************************************
/* Token buffering state
/**********************************************************
 */

/**
 * First segment, for contents this buffer has
 */
protected Segment _first;

/**
 * Last segment of this buffer, one that is used
 * for appending more tokens
 */
protected Segment _last;

/**
 * Offset within last segment,
 */
protected int _appendOffset;

/*
/**********************************************************
/* Output state
/**********************************************************
 */
```

```
protected JsonWriteContext _writeContext;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

/**
 * @param codec Object codec to use for stream-based object
 *   conversion through parser/generator interfaces. If null,
 *   such methods can not be used.
 */
public TokenBuffer(ObjectCodec codec)
{
   _objectCodec = codec;
   _generatorFeatures = DEFAULT_PARSER_FEATURES;
   _writeContext = JsonWriteContext.createRootContext();
   // at first we have just one segment
   _first = _last = new Segment();
   _appendOffset = 0;
}

/**
 * Method used to create a {@link JsonParser} that can read contents
 * stored in this buffer. Will use default <code>_objectCodec</code> for
 * object conversions.
 *<p>
 * Note: instances are not synchronized, that is, they are not thread-safe
 * if there are concurrent appends to the underlying buffer.
 *
 * @return Parser that can be used for reading contents stored in this buffer
 */
public JsonParser asParser()
{
   return asParser(_objectCodec);
}

/**
 * Method used to create a {@link JsonParser} that can read contents
 * stored in this buffer.
 *<p>
 * Note: instances are not synchronized, that is, they are not thread-safe
 * if there are concurrent appends to the underlying buffer.
 *
 * @param codec Object codec to use for stream-based object
 *   conversion through parser/generator interfaces. If null,
 *   such methods can not be used.
```

```
     *
     * @return Parser that can be used for reading contents stored in this buffer
     */
    public JsonParser asParser(ObjectCodec codec)
    {
        return new Parser(_first, codec);
    }

    /**
     * @param src Parser to use for accessing source information
     *    like location, configured codec
     */
    public JsonParser asParser(JsonParser src)
    {
        Parser p = new Parser(_first, src.getCodec());
        p.setLocation(src.getTokenLocation());
        return p;
    }

    /*
    /**********************************************************
    /* Other custom methods not needed for implementing interfaces
    /**********************************************************
     */

    /**
     * Helper method that will write all contents of this buffer
     * using given {@link JsonGenerator}.
     *<p>
     * Note: this method would be enough to implement
     * <code>JsonSerializer</code>  for <code>TokenBuffer</code> type;
     * but we can not have upwards
     * references (from core to mapper package); and as such we also
     * can not take second argument.
     */
    public void serialize(JsonGenerator jgen)
        throws IOException, JsonGenerationException
    {
        Segment segment = _first;
        int ptr = -1;

        while (true) {
            if (++ptr >= Segment.TOKENS_PER_SEGMENT) {
                ptr = 0;
                segment = segment.next();
                if (segment == null) break;
            }
            JsonToken t = segment.type(ptr);
```

```
if (t == null) break;

// Note: copied from 'copyCurrentEvent'...
switch (t) {
case START_OBJECT:
   jgen.writeStartObject();
   break;
case END_OBJECT:
   jgen.writeEndObject();
   break;
case START_ARRAY:
   jgen.writeStartArray();
   break;
case END_ARRAY:
   jgen.writeEndArray();
   break;
case FIELD_NAME:
{
   // 13-Dec-2010, tatu: Maybe we should start using different type tokens to reduce casting?
   Object ob = segment.get(ptr);
   if (ob instanceof SerializableString) {
      jgen.writeFieldName((SerializableString) ob);
   } else {
      jgen.writeFieldName((String) ob);
   }
}
   break;
case VALUE_STRING:
   {
      Object ob = segment.get(ptr);
      if (ob instanceof SerializableString) {
         jgen.writeString((SerializableString) ob);
      } else {
         jgen.writeString((String) ob);
      }
   }
   break;
case VALUE_NUMBER_INT:
   {
      Number n = (Number) segment.get(ptr);
      if (n instanceof BigInteger) {
         jgen.writeNumber((BigInteger) n);
      } else if (n instanceof Long) {
         jgen.writeNumber(n.longValue());
      } else {
         jgen.writeNumber(n.intValue());
      }
   }
```

```
            break;
        case VALUE_NUMBER_FLOAT:
            {
                Object n = segment.get(ptr);
                if (n instanceof BigDecimal) {
                    jgen.writeNumber((BigDecimal) n);
                } else if (n instanceof Float) {
                    jgen.writeNumber(((Float) n).floatValue());
                } else if (n instanceof Double) {
                    jgen.writeNumber(((Double) n).doubleValue());
                } else if (n == null) {
                    jgen.writeNull();
                } else if (n instanceof String) {
                    jgen.writeNumber((String) n);
                } else {
                    throw new JsonGenerationException("Unrecognized value type for VALUE_NUMBER_FLOAT:
"+n.getClass().getName()+", can not serialize");
                }
            }
            break;
        case VALUE_TRUE:
            jgen.writeBoolean(true);
            break;
        case VALUE_FALSE:
            jgen.writeBoolean(false);
            break;
        case VALUE_NULL:
            jgen.writeNull();
            break;
        case VALUE_EMBEDDED_OBJECT:
            jgen.writeObject(segment.get(ptr));
            break;
        default:
            throw new RuntimeException("Internal error: should never end up through this code path");
        }
    }
  }

  @Override
  public String toString()
  {
    // Let's print up to 100 first tokens...
    final int MAX_COUNT = 100;

    StringBuilder sb = new StringBuilder();
    sb.append("[TokenBuffer: ");
    JsonParser jp = asParser();
    int count = 0;
```

```
    while (true) {
        JsonToken t;
        try {
            t = jp.nextToken();
        } catch (IOException ioe) { // should never occur
            throw new IllegalStateException(ioe);
        }
        if (t == null) break;
        if (count < MAX_COUNT) {
            if (count > 0) {
                sb.append(", ");
            }
            sb.append(t.toString());
        }
        ++count;
    }

    if (count >= MAX_COUNT) {
        sb.append(" ... (truncated ").append(count-MAX_COUNT).append(" entries)");
    }
    sb.append(']');
    return sb.toString();
}

/*
/**********************************************************
/* JsonGenerator implementation: configuration
/**********************************************************
 */

@Override
public JsonGenerator enable(Feature f) {
    _generatorFeatures |= f.getMask();
    return this;
}

@Override
public JsonGenerator disable(Feature f) {
    _generatorFeatures &= ~f.getMask();
    return this;
}

//public JsonGenerator configure(Feature f, boolean state) { }

@Override
public boolean isEnabled(Feature f) {
    return (_generatorFeatures & f.getMask()) != 0;
```

```
}

@Override
public JsonGenerator useDefaultPrettyPrinter() {
   // No-op: we don't indent
   return this;
}

@Override
public JsonGenerator setCodec(ObjectCodec oc) {
   _objectCodec = oc;
   return this;
}

@Override
public ObjectCodec getCodec() { return _objectCodec; }

@Override
public final JsonWriteContext getOutputContext() { return _writeContext; }

/*
/**********************************************************
/* JsonGenerator implementation: low-level output handling
/**********************************************************
 */

@Override
public void flush() throws IOException { /* NOP */ }

@Override
public void close() throws IOException {
   _closed = true;
}

@Override
public boolean isClosed() { return _closed; }

/*
/**********************************************************
/* JsonGenerator implementation: write methods, structural
/**********************************************************
 */

@Override
public final void writeStartArray()
   throws IOException, JsonGenerationException
{
   _append(JsonToken.START_ARRAY);
```

```
    _writeContext = _writeContext.createChildArrayContext();
}

@Override
public final void writeEndArray()
    throws IOException, JsonGenerationException
{
    _append(JsonToken.END_ARRAY);
    // Let's allow unbalanced tho... i.e. not run out of root level, ever
    JsonWriteContext c = _writeContext.getParent();
    if (c != null) {
        _writeContext = c;
    }
}

@Override
public final void writeStartObject()
    throws IOException, JsonGenerationException
{
    _append(JsonToken.START_OBJECT);
    _writeContext = _writeContext.createChildObjectContext();
}

@Override
public final void writeEndObject()
    throws IOException, JsonGenerationException
{
    _append(JsonToken.END_OBJECT);
    // Let's allow unbalanced tho... i.e. not run out of root level, ever
    JsonWriteContext c = _writeContext.getParent();
    if (c != null) {
        _writeContext = c;
    }
}

@Override
public final void writeFieldName(String name)
    throws IOException, JsonGenerationException
{
    _append(JsonToken.FIELD_NAME, name);
    _writeContext.writeFieldName(name);
}

@Override
public void writeFieldName(SerializableString name)
    throws IOException, JsonGenerationException
{
    _append(JsonToken.FIELD_NAME, name);
```

```
    _writeContext.writeFieldName(name.getValue());
}

@Override
public void writeFieldName(SerializedString name)
    throws IOException, JsonGenerationException
{
    _append(JsonToken.FIELD_NAME, name);
    _writeContext.writeFieldName(name.getValue());
}

/*
/**********************************************************
/* JsonGenerator implementation: write methods, textual
/**********************************************************
 */

@Override
public void writeString(String text) throws IOException,JsonGenerationException {
    if (text == null) {
        writeNull();
    } else {
        _append(JsonToken.VALUE_STRING, text);
    }
}

@Override
public void writeString(char[] text, int offset, int len) throws IOException, JsonGenerationException {
    writeString(new String(text, offset, len));
}

@Override
public void writeString(SerializableString text) throws IOException, JsonGenerationException {
    if (text == null) {
        writeNull();
    } else {
        _append(JsonToken.VALUE_STRING, text);
    }
}

@Override
public void writeRawUTF8String(byte[] text, int offset, int length)
    throws IOException, JsonGenerationException
{
    // could add support for buffering if we really want it...
    _reportUnsupportedOperation();
}
```

```
@Override
public void writeUTF8String(byte[] text, int offset, int length)
   throws IOException, JsonGenerationException
{
   // could add support for buffering if we really want it...
   _reportUnsupportedOperation();
}

@Override
public void writeRaw(String text) throws IOException, JsonGenerationException {
   _reportUnsupportedOperation();
}

@Override
public void writeRaw(String text, int offset, int len) throws IOException, JsonGenerationException {
   _reportUnsupportedOperation();
}

@Override
public void writeRaw(char[] text, int offset, int len) throws IOException, JsonGenerationException {
   _reportUnsupportedOperation();
}

@Override
public void writeRaw(char c) throws IOException, JsonGenerationException {
   _reportUnsupportedOperation();
}

@Override
public void writeRawValue(String text) throws IOException, JsonGenerationException {
   _reportUnsupportedOperation();
}

@Override
public void writeRawValue(String text, int offset, int len) throws IOException, JsonGenerationException {
   _reportUnsupportedOperation();
}

@Override
public void writeRawValue(char[] text, int offset, int len) throws IOException, JsonGenerationException {
   _reportUnsupportedOperation();
}

/*
/**********************************************************
/* JsonGenerator implementation: write methods, primitive types
/**********************************************************
 */
```

```
@Override
public void writeNumber(int i) throws IOException, JsonGenerationException {
   _append(JsonToken.VALUE_NUMBER_INT, Integer.valueOf(i));
}

@Override
public void writeNumber(long l) throws IOException, JsonGenerationException {
   _append(JsonToken.VALUE_NUMBER_INT, Long.valueOf(l));
}

@Override
public void writeNumber(double d) throws IOException,JsonGenerationException {
   _append(JsonToken.VALUE_NUMBER_FLOAT, Double.valueOf(d));
}

@Override
public void writeNumber(float f) throws IOException, JsonGenerationException {
   _append(JsonToken.VALUE_NUMBER_FLOAT, Float.valueOf(f));
}

@Override
public void writeNumber(BigDecimal dec) throws IOException,JsonGenerationException {
   if (dec == null) {
      writeNull();
   } else {
      _append(JsonToken.VALUE_NUMBER_FLOAT, dec);
   }
}

@Override
public void writeNumber(BigInteger v) throws IOException, JsonGenerationException {
   if (v == null) {
      writeNull();
   } else {
      _append(JsonToken.VALUE_NUMBER_INT, v);
   }
}

@Override
public void writeNumber(String encodedValue) throws IOException, JsonGenerationException {
   /* 03-Dec-2010, tatu: related to [JACKSON-423], should try to keep as numeric
    *   identity as long as possible
    */
   _append(JsonToken.VALUE_NUMBER_FLOAT, encodedValue);
}

@Override
```

```
public void writeBoolean(boolean state) throws IOException,JsonGenerationException {
   _append(state ? JsonToken.VALUE_TRUE : JsonToken.VALUE_FALSE);
}

@Override
public void writeNull() throws IOException, JsonGenerationException {
   _append(JsonToken.VALUE_NULL);
}

/*
/**********************************************************
/* JsonGenerator implementation: write methods for POJOs/trees
/**********************************************************
 */

@Override
public void writeObject(Object value)
   throws IOException, JsonProcessingException
{
   // embedded means that no conversions should be done...
   _append(JsonToken.VALUE_EMBEDDED_OBJECT, value);
}

@Override
public void writeTree(JsonNode rootNode)
   throws IOException, JsonProcessingException
{
   /* 31-Dec-2009, tatu: no need to convert trees either is there?
    *  (note: may need to re-evaluate at some point)
    */
   _append(JsonToken.VALUE_EMBEDDED_OBJECT, rootNode);
}

/*
/**********************************************************
/* JsonGenerator implementation; binary
/**********************************************************
 */

@Override
public void writeBinary(Base64Variant b64variant, byte[] data, int offset, int len)
   throws IOException, JsonGenerationException
{
   /* 31-Dec-2009, tatu: can do this using multiple alternatives; but for
    *   now, let's try to limit number of conversions.
    *   The only (?) tricky thing is that of whether to preserve variant,
    *   seems pointless, so let's not worry about it unless there's some
    *   compelling reason to.
```

```
     */
    byte[] copy = new byte[len];
    System.arraycopy(data, offset, copy, 0, len);
    writeObject(copy);
}

/*
/**********************************************************
/* JsonGenerator implementation; pass-through copy
/**********************************************************
 */

@Override
public void copyCurrentEvent(JsonParser jp) throws IOException, JsonProcessingException
{
    switch (jp.getCurrentToken()) {
    case START_OBJECT:
        writeStartObject();
        break;
    case END_OBJECT:
        writeEndObject();
        break;
    case START_ARRAY:
        writeStartArray();
        break;
    case END_ARRAY:
        writeEndArray();
        break;
    case FIELD_NAME:
        writeFieldName(jp.getCurrentName());
        break;
    case VALUE_STRING:
        if (jp.hasTextCharacters()) {
            writeString(jp.getTextCharacters(), jp.getTextOffset(), jp.getTextLength());
        } else {
            writeString(jp.getText());
        }
        break;
    case VALUE_NUMBER_INT:
        switch (jp.getNumberType()) {
        case INT:
            writeNumber(jp.getIntValue());
            break;
        case BIG_INTEGER:
            writeNumber(jp.getBigIntegerValue());
            break;
        default:
            writeNumber(jp.getLongValue());
```

```
            }
            break;
        case VALUE_NUMBER_FLOAT:
            switch (jp.getNumberType()) {
            case BIG_DECIMAL:
                writeNumber(jp.getDecimalValue());
                break;
            case FLOAT:
                writeNumber(jp.getFloatValue());
                break;
            default:
                writeNumber(jp.getDoubleValue());
            }
            break;
        case VALUE_TRUE:
            writeBoolean(true);
            break;
        case VALUE_FALSE:
            writeBoolean(false);
            break;
        case VALUE_NULL:
            writeNull();
            break;
        case VALUE_EMBEDDED_OBJECT:
            writeObject(jp.getEmbeddedObject());
            break;
        default:
            throw new RuntimeException("Internal error: should never end up through this code path");
        }
    }

    @Override
    public void copyCurrentStructure(JsonParser jp) throws IOException, JsonProcessingException {
        JsonToken t = jp.getCurrentToken();

        // Let's handle field-name separately first
        if (t == JsonToken.FIELD_NAME) {
            writeFieldName(jp.getCurrentName());
            t = jp.nextToken();
            // fall-through to copy the associated value
        }

        switch (t) {
        case START_ARRAY:
            writeStartArray();
            while (jp.nextToken() != JsonToken.END_ARRAY) {
                copyCurrentStructure(jp);
            }
```

```
            writeEndArray();
            break;
        case START_OBJECT:
            writeStartObject();
            while (jp.nextToken() != JsonToken.END_OBJECT) {
                copyCurrentStructure(jp);
            }
            writeEndObject();
            break;
        default: // others are simple:
            copyCurrentEvent(jp);
        }
    }

    /*
    /**********************************************************
    /* Internal methods
    /**********************************************************
     */
    protected final void _append(JsonToken type) {
        Segment next = _last.append(_appendOffset, type);
        if (next == null) {
            ++_appendOffset;
        } else {
            _last = next;
            _appendOffset = 1; // since we added first at 0
        }
    }

    protected final void _append(JsonToken type, Object value) {
        Segment next = _last.append(_appendOffset, type, value);
        if (next == null) {
            ++_appendOffset;
        } else {
            _last = next;
            _appendOffset = 1;
        }
    }

    protected void _reportUnsupportedOperation() {
        throw new UnsupportedOperationException("Called operation not supported for TokenBuffer");
    }

    /*
    /**********************************************************
    /* Supporting classes
    /**********************************************************
     */
```

```
protected final static class Parser
    extends JsonParserMinimalBase
{
    protected ObjectCodec _codec;

    /*
    /**********************************************************
    /* Parsing state
    /**********************************************************
     */

    /**
     * Currently active segment
     */
    protected Segment _segment;

    /**
     * Pointer to current token within current segment
     */
    protected int _segmentPtr;

    /**
     * Information about parser context, context in which
     * the next token is to be parsed (root, array, object).
     */
    protected JsonReadContext _parsingContext;

    protected boolean _closed;

    protected transient ByteArrayBuilder _byteBuilder;

    protected JsonLocation _location = null;

    /*
    /**********************************************************
    /* Construction, init
    /**********************************************************
     */

    public Parser(Segment firstSeg, ObjectCodec codec)
    {
        super(0);
        _segment = firstSeg;
        _segmentPtr = -1; // not yet read
        _codec = codec;
        _parsingContext = JsonReadContext.createRootContext(-1, -1);
    }
```

```
public void setLocation(JsonLocation l) {
   _location = l;
}

@Override
public ObjectCodec getCodec() { return _codec; }

@Override
public void setCodec(ObjectCodec c) { _codec = c; }

/*
/**********************************************************
/* Extended API beyond JsonParser
/**********************************************************
 */

public JsonToken peekNextToken()
   throws IOException, JsonParseException
{
   // closed? nothing more to peek, either
   if (_closed) return null;
   Segment seg = _segment;
   int ptr = _segmentPtr+1;
   if (ptr >= Segment.TOKENS_PER_SEGMENT) {
      ptr = 0;
      seg = (seg == null) ? null : seg.next();
   }
   return (seg == null) ? null : seg.type(ptr);
}

/*
/**********************************************************
/* Closeable implementation
/**********************************************************
 */

@Override
public void close() throws IOException {
   if (!_closed) {
      _closed = true;
   }
}

/*
/**********************************************************
/* Public API, traversal
/**********************************************************
```

```
 */

@Override
public JsonToken nextToken() throws IOException, JsonParseException
{
   // If we are closed, nothing more to do
   if (_closed || (_segment == null)) return null;

   // Ok, then: any more tokens?
   if (++_segmentPtr >= Segment.TOKENS_PER_SEGMENT) {
      _segmentPtr = 0;
      _segment = _segment.next();
      if (_segment == null) {
         return null;
      }
   }
   _currToken = _segment.type(_segmentPtr);
   // Field name? Need to update context
   if (_currToken == JsonToken.FIELD_NAME) {
      Object ob = _currentObject();
      String name = (ob instanceof String) ? ((String) ob) : ob.toString();
      _parsingContext.setCurrentName(name);
   } else if (_currToken == JsonToken.START_OBJECT) {
      _parsingContext = _parsingContext.createChildObjectContext(-1, -1);
   } else if (_currToken == JsonToken.START_ARRAY) {
      _parsingContext = _parsingContext.createChildArrayContext(-1, -1);
   } else if (_currToken == JsonToken.END_OBJECT
         || _currToken == JsonToken.END_ARRAY) {
      // Closing JSON Object/Array? Close matching context
      _parsingContext = _parsingContext.getParent();
      // but allow unbalanced cases too (more close markers)
      if (_parsingContext == null) {
         _parsingContext = JsonReadContext.createRootContext(-1, -1);
      }
   }
   return _currToken;
}

@Override
public boolean isClosed() { return _closed; }

/*
/**********************************************************
/* Public API, token accessors
/**********************************************************
 */

@Override
```

```
public JsonStreamContext getParsingContext() { return _parsingContext; }

@Override
public JsonLocation getTokenLocation() { return getCurrentLocation(); }

@Override
public JsonLocation getCurrentLocation() {
   return (_location == null) ? JsonLocation.NA : _location;
}

@Override
public String getCurrentName() { return _parsingContext.getCurrentName(); }

/*
/**********************************************************
/* Public API, access to token information, text
/**********************************************************
 */

@Override
public String getText()
{
   // common cases first:
   if (_currToken == JsonToken.VALUE_STRING
         || _currToken == JsonToken.FIELD_NAME) {
      Object ob = _currentObject();
      if (ob instanceof String) {
         return (String) ob;
      }
      return (ob == null) ? null : ob.toString();
   }
   if (_currToken == null) {
      return null;
   }
   switch (_currToken) {
   case VALUE_NUMBER_INT:
   case VALUE_NUMBER_FLOAT:
      Object ob = _currentObject();
      return (ob == null) ? null : ob.toString();
   }
   return _currToken.asString();
}

@Override
public char[] getTextCharacters() {
   String str = getText();
   return (str == null) ? null : str.toCharArray();
}
```

```
@Override
public int getTextLength() {
   String str = getText();
   return (str == null) ? 0 : str.length();
}

@Override
public int getTextOffset() { return 0; }

@Override
public boolean hasTextCharacters() {
   // We never have raw buffer available, so:
   return false;
}

/*
/**********************************************************
/* Public API, access to token information, numeric
/**********************************************************
 */

@Override
public BigInteger getBigIntegerValue() throws IOException, JsonParseException
{
   Number n = getNumberValue();
   if (n instanceof BigInteger) {
      return (BigInteger) n;
   }
   switch (getNumberType()) {
   case BIG_DECIMAL:
      return ((BigDecimal) n).toBigInteger();
   }
   // int/long is simple, but let's also just truncate float/double:
   return BigInteger.valueOf(n.longValue());
}

@Override
public BigDecimal getDecimalValue() throws IOException, JsonParseException
{
   Number n = getNumberValue();
   if (n instanceof BigDecimal) {
      return (BigDecimal) n;
   }
   switch (getNumberType()) {
   case INT:
   case LONG:
      return BigDecimal.valueOf(n.longValue());
```

```
    case BIG_INTEGER:
        return new BigDecimal((BigInteger) n);
    }
    // float or double
    return BigDecimal.valueOf(n.doubleValue());
}

@Override
public double getDoubleValue() throws IOException, JsonParseException {
    return getNumberValue().doubleValue();
}

@Override
public float getFloatValue() throws IOException, JsonParseException {
    return getNumberValue().floatValue();
}

@Override
public int getIntValue() throws IOException, JsonParseException
{
    // optimize common case:
    if (_currToken == JsonToken.VALUE_NUMBER_INT) {
        return ((Number) _currentObject()).intValue();
    }
    return getNumberValue().intValue();
}

@Override
public long getLongValue() throws IOException, JsonParseException {
    return getNumberValue().longValue();
}

@Override
public NumberType getNumberType() throws IOException, JsonParseException
{
    Number n = getNumberValue();
    if (n instanceof Integer) return NumberType.INT;
    if (n instanceof Long) return NumberType.LONG;
    if (n instanceof Double) return NumberType.DOUBLE;
    if (n instanceof BigDecimal) return NumberType.BIG_DECIMAL;
    if (n instanceof Float) return NumberType.FLOAT;
    if (n instanceof BigInteger) return NumberType.BIG_INTEGER;
    return null;
}

@Override
public final Number getNumberValue() throws IOException, JsonParseException {
    _checkIsNumber();
```

```
        return (Number) _currentObject();
    }

    /*
    /**********************************************************
    /* Public API, access to token information, other
    /**********************************************************
     */

    @Override
    public Object getEmbeddedObject()
    {
        if (_currToken == JsonToken.VALUE_EMBEDDED_OBJECT) {
            return _currentObject();
        }
        return null;
    }

    @Override
    public byte[] getBinaryValue(Base64Variant b64variant) throws IOException, JsonParseException
    {
        // First: maybe we some special types?
        if (_currToken == JsonToken.VALUE_EMBEDDED_OBJECT) {
            // Embedded byte array would work nicely...
            Object ob = _currentObject();
            if (ob instanceof byte[]) {
                return (byte[]) ob;
            }
            // fall through to error case
        }
        if (_currToken != JsonToken.VALUE_STRING) {
            throw _constructError("Current token ("+_currToken+") not VALUE_STRING (or
VALUE_EMBEDDED_OBJECT with byte[]), can not access as binary");
        }
        final String str = getText();
        if (str == null) {
            return null;
        }
        ByteArrayBuilder builder = _byteBuilder;
        if (builder == null) {
            _byteBuilder = builder = new ByteArrayBuilder(100);
        }
        _decodeBase64(str, builder, b64variant);
        return builder.toByteArray();
    }

    /*
    /**********************************************************
```

```
/* Internal methods
/**********************************************************
 */

protected void _decodeBase64(String str, ByteArrayBuilder builder, Base64Variant b64variant)
   throws IOException, JsonParseException
{
   int ptr = 0;
   int len = str.length();

   main_loop:
   while (ptr < len) {
      // first, we'll skip preceding white space, if any
      char ch;
      do {
         ch = str.charAt(ptr++);
         if (ptr >= len) {
            break main_loop;
         }
      } while (ch <= INT_SPACE);
      int bits = b64variant.decodeBase64Char(ch);
      if (bits < 0) {
         _reportInvalidBase64(b64variant, ch, 0, null);
      }
      int decodedData = bits;
      // then second base64 char; can't get padding yet, nor ws
      if (ptr >= len) {
         _reportBase64EOF();
      }
      ch = str.charAt(ptr++);
      bits = b64variant.decodeBase64Char(ch);
      if (bits < 0) {
         _reportInvalidBase64(b64variant, ch, 1, null);
      }
      decodedData = (decodedData << 6) | bits;
      // third base64 char; can be padding, but not ws
      if (ptr >= len) {
         _reportBase64EOF();
      }
      ch = str.charAt(ptr++);
      bits = b64variant.decodeBase64Char(ch);

      // First branch: can get padding (-> 1 byte)
      if (bits < 0) {
         if (bits != Base64Variant.BASE64_VALUE_PADDING) {
            _reportInvalidBase64(b64variant, ch, 2, null);
         }
         // Ok, must get padding
```

```
                if (ptr >= len) {
                    _reportBase64EOF();
                }
                ch = str.charAt(ptr++);
                if (!b64variant.usesPaddingChar(ch)) {
                    _reportInvalidBase64(b64variant, ch, 3, "expected padding character
'"+b64variant.getPaddingChar()+"'");
                }
                // Got 12 bits, only need 8, need to shift
                decodedData >>= 4;
                builder.append(decodedData);
                continue;
            }
            // Nope, 2 or 3 bytes
            decodedData = (decodedData << 6) | bits;
            // fourth and last base64 char; can be padding, but not ws
            if (ptr >= len) {
                _reportBase64EOF();
            }
            ch = str.charAt(ptr++);
            bits = b64variant.decodeBase64Char(ch);
            if (bits < 0) {
                if (bits != Base64Variant.BASE64_VALUE_PADDING) {
                    _reportInvalidBase64(b64variant, ch, 3, null);
                }
                decodedData >>= 2;
                builder.appendTwoBytes(decodedData);
            } else {
                // otherwise, our triple is now complete
                decodedData = (decodedData << 6) | bits;
                builder.appendThreeBytes(decodedData);
            }
        }
    }

    protected final Object _currentObject() {
        return _segment.get(_segmentPtr);
    }

    protected final void _checkIsNumber() throws JsonParseException
    {
        if (_currToken == null || !_currToken.isNumeric()) {
            throw _constructError("Current token ("+_currToken+") not numeric, can not use numeric value
accessors");
        }
    }

    /**
```

```
     * @param bindex Relative index within base64 character unit; between 0
     *   and 3 (as unit has exactly 4 characters)
     */
    protected void _reportInvalidBase64(Base64Variant b64variant, char ch, int bindex, String msg)
        throws JsonParseException
    {
        String base;
        if (ch <= INT_SPACE) {
            base = "Illegal white space character (code 0x"+Integer.toHexString(ch)+") as character #"+(bindex+1)+"
of 4-char base64 unit: can only used between units";
        } else if (b64variant.usesPaddingChar(ch)) {
            base = "Unexpected padding character ('"+b64variant.getPaddingChar()+"') as character #"+(bindex+1)+"
of 4-char base64 unit: padding only legal as 3rd or 4th character";
        } else if (!Character.isDefined(ch) || Character.isISOControl(ch)) {
            // Not sure if we can really get here... ? (most illegal xml chars are caught at lower level)
            base = "Illegal character (code 0x"+Integer.toHexString(ch)+") in base64 content";
        } else {
            base = "Illegal character '"+ch+"' (code 0x"+Integer.toHexString(ch)+") in base64 content";
        }
        if (msg != null) {
            base = base + ": " + msg;
        }
        throw _constructError(base);
    }

    protected void _reportBase64EOF() throws JsonParseException {
        throw _constructError("Unexpected end-of-String in base64 content");
    }

    @Override
    protected void _handleEOF() throws JsonParseException {
        _throwInternal();
    }
}

/**
 * Individual segment of TokenBuffer that can store up to 16 tokens
 * (limited by 4 bits per token type marker requirement).
 * Current implementation uses fixed length array; could alternatively
 * use 16 distinct fields and switch statement (slightly more efficient
 * storage, slightly slower access)
 */
protected final static class Segment
{
    public final static int TOKENS_PER_SEGMENT = 16;

    /**
     * Static array used for fast conversion between token markers and
```

```
 * matching {@link JsonToken} instances
 */
private final static JsonToken[] TOKEN_TYPES_BY_INDEX;
static {
   // ... here we know that there are <= 16 values in JsonToken enum
   TOKEN_TYPES_BY_INDEX = new JsonToken[16];
   JsonToken[] t = JsonToken.values();
   System.arraycopy(t, 1, TOKEN_TYPES_BY_INDEX, 1, Math.min(15, t.length - 1));
}

// // // Linking

protected Segment _next;

// // // State

/**
 * Bit field used to store types of buffered tokens; 4 bits per token.
 * Value 0 is reserved for "not in use"
 */
protected long _tokenTypes;


// Actual tokens

protected final Object[] _tokens = new Object[TOKENS_PER_SEGMENT];

public Segment() { }

// // // Accessors

public JsonToken type(int index)
{
   long l = _tokenTypes;
   if (index > 0) {
      l >>= (index << 2);
   }
   int ix = ((int) l) & 0xF;
   return TOKEN_TYPES_BY_INDEX[ix];
}

public Object get(int index) {
   return _tokens[index];
}

public Segment next() { return _next; }

// // // Mutators
```

```
    public Segment append(int index, JsonToken tokenType)
    {
        if (index < TOKENS_PER_SEGMENT) {
            set(index, tokenType);
            return null;
        }
        _next = new Segment();
        _next.set(0, tokenType);
        return _next;
    }

    public Segment append(int index, JsonToken tokenType, Object value)
    {
        if (index < TOKENS_PER_SEGMENT) {
            set(index, tokenType, value);
            return null;
        }
        _next = new Segment();
        _next.set(0, tokenType, value);
        return _next;
    }

    public void set(int index, JsonToken tokenType)
    {
        long typeCode = tokenType.ordinal();
        /* Assumption here is that there are no overwrites, just appends;
         * and so no masking is needed
         */
        if (index > 0) {
            typeCode <<= (index << 2);
        }
        _tokenTypes |= typeCode;
    }

    public void set(int index, JsonToken tokenType, Object value)
    {
        _tokens[index] = value;
        long typeCode = tokenType.ordinal();
        /* Assumption here is that there are no overwrites, just appends;
         * and so no masking is needed
         */
        if (index > 0) {
            typeCode <<= (index << 2);
        }
        _tokenTypes |= typeCode;
    }
}
```

```
}
package org.codehaus.jackson.util;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;

/**
* Helper class that implements
* <a href="http://en.wikipedia.org/wiki/Delegation_pattern">delegation pattern</a> for {@link JsonParser},
* to allow for simple overridability of basic parsing functionality.
* The idea is that any functionality to be modified can be simply
* overridden; and anything else will be delegated by default.
*
* @since 1.4
*/
public class JsonParserDelegate extends JsonParser
{
  /**
   * Delegate object that method calls are delegated to.
   */
  protected JsonParser delegate;

  public JsonParserDelegate(JsonParser d) {
     delegate = d;
  }

  /*
  /**********************************************************
  /* Public API, configuration
  /**********************************************************
   */

  @Override
  public void setCodec(ObjectCodec c) {
     delegate.setCodec(c);
  }

  @Override
  public ObjectCodec getCodec() {
     return delegate.getCodec();
  }

  @Override
  public JsonParser enable(Feature f) {
     delegate.enable(f);
```

```
    return this;
}

@Override
public JsonParser disable(Feature f) {
    delegate.disable(f);
    return this;
}

@Override
public boolean isEnabled(Feature f) {
    return delegate.isEnabled(f);
}

/*
/**********************************************************
/* Closeable impl
/**********************************************************
 */

@Override
public void close() throws IOException {
    delegate.close();
}

@Override
public boolean isClosed() {
    return delegate.isClosed();
}

/*
/**********************************************************
/* Public API, token accessors
/**********************************************************
 */

@Override
public JsonToken getCurrentToken() {
    return delegate.getCurrentToken();
}

@Override
public boolean hasCurrentToken() {
    return delegate.hasCurrentToken();
}

@Override
public void clearCurrentToken() {
```

```
    delegate.clearCurrentToken();
}

@Override
public String getCurrentName() throws IOException, JsonParseException {
    return delegate.getCurrentName();
}

@Override
public JsonLocation getCurrentLocation() {
    return delegate.getCurrentLocation();
}

@Override
public JsonToken getLastClearedToken() {
    return delegate.getLastClearedToken();
}

@Override
public JsonStreamContext getParsingContext() {
    return delegate.getParsingContext();
}

/*
/**********************************************************
/* Public API, access to token information, text
/**********************************************************
 */

@Override
public String getText() throws IOException, JsonParseException {
    return delegate.getText();
}

@Override
public char[] getTextCharacters() throws IOException, JsonParseException {
    return delegate.getTextCharacters();
}

@Override
public int getTextLength() throws IOException, JsonParseException {
    return delegate.getTextLength();
}

@Override
public int getTextOffset() throws IOException, JsonParseException {
    return delegate.getTextOffset();
}
```

```
/*
/************************************************************
/* Public API, access to token information, numeric
/************************************************************
 */

@Override
public BigInteger getBigIntegerValue() throws IOException,JsonParseException {
   return delegate.getBigIntegerValue();
}

@Override
public byte getByteValue() throws IOException, JsonParseException {
   return delegate.getByteValue();
}

@Override
public short getShortValue() throws IOException, JsonParseException {
   return delegate.getShortValue();
}

@Override
public BigDecimal getDecimalValue() throws IOException, JsonParseException {
   return delegate.getDecimalValue();
}

@Override
public double getDoubleValue() throws IOException, JsonParseException {
   return delegate.getDoubleValue();
}

@Override
public float getFloatValue() throws IOException, JsonParseException {
   return delegate.getFloatValue();
}

@Override
public int getIntValue() throws IOException, JsonParseException {
   return delegate.getIntValue();
}

@Override
public long getLongValue() throws IOException, JsonParseException {
   return delegate.getLongValue();
}
```

```
  @Override
  public NumberType getNumberType() throws IOException, JsonParseException {
    return delegate.getNumberType();
  }

  @Override
  public Number getNumberValue() throws IOException, JsonParseException {
    return delegate.getNumberValue();
  }

  @Override
  public byte[] getBinaryValue(Base64Variant b64variant) throws IOException, JsonParseException {
    return delegate.getBinaryValue(b64variant);
  }

  @Override
  public JsonLocation getTokenLocation() {
    return delegate.getTokenLocation();
  }

  @Override
  public JsonToken nextToken() throws IOException, JsonParseException {
    return delegate.nextToken();
  }

  @Override
  public JsonParser skipChildren() throws IOException, JsonParseException {
    delegate.skipChildren();
    // NOTE: must NOT delegate this method to delegate, needs to be self-reference for chaining
    return this;
  }
}
package org.codehaus.jackson.util;

import java.io.*;
import java.util.Arrays;

import org.codehaus.jackson.*;
import org.codehaus.jackson.impl.Indenter;

/**
* Default {@link PrettyPrinter} implementation that uses 2-space
* indentation with platform-default linefeeds.
* Usually this class is not instantiated directly, but instead
* method {@link JsonGenerator#useDefaultPrettyPrinter} is
* used, which will use an instance of this class for operation.
*/
public class DefaultPrettyPrinter
```

```
  implements PrettyPrinter
{
  // // // Config, indentation

  /**
   * By default, let's use only spaces to separate array values.
   */
  protected Indenter _arrayIndenter = new FixedSpaceIndenter();

  /**
   * By default, let's use linefeed-adding indenter for separate
   * object entries. We'll further configure indenter to use
   * system-specific linefeeds, and 2 spaces per level (as opposed to,
   * say, single tabs)
   */
  protected Indenter _objectIndenter = new Lf2SpacesIndenter();

  // // // Config, other white space configuration

  /**
   * By default we will add spaces around colons used to
   * separate object fields and values.
   * If disabled, will not use spaces around colon.
   */
  protected boolean _spacesInObjectEntries = true;

  // // // State:

  /**
   * Number of open levels of nesting. Used to determine amount of
   * indentation to use.
   */
  protected int _nesting = 0;

  /*
  /**********************************************************
  /* Life-cycle (construct, configure)
  /**********************************************************
  */

  public DefaultPrettyPrinter() { }

  public void indentArraysWith(Indenter i)
  {
     _arrayIndenter = (i == null) ? new NopIndenter() : i;
  }

  public void indentObjectsWith(Indenter i)
```

```
{
    _objectIndenter = (i == null) ? new NopIndenter() : i;
}

public void spacesInObjectEntries(boolean b) { _spacesInObjectEntries = b; }

/*
/**********************************************************
/* PrettyPrinter impl
/**********************************************************
 */

public void writeRootValueSeparator(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    jg.writeRaw(' ');
}

public void writeStartObject(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    jg.writeRaw('{');
    if (!_objectIndenter.isInline()) {
        ++_nesting;
    }
}

public void beforeObjectEntries(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    _objectIndenter.writeIndentation(jg, _nesting);
}

/**
 * Method called after an object field has been output, but
 * before the value is output.
 *<p>
 * Default handling (without pretty-printing) will output a single
 * colon to separate the two. Pretty-printer is
 * to output a colon as well, but can surround that with other
 * (white-space) decoration.
 */
public void writeObjectFieldValueSeparator(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    if (_spacesInObjectEntries) {
        jg.writeRaw(" : ");
    } else {
```

```
        jg.writeRaw(':');
    }
}

/**
 * Method called after an object entry (field:value) has been completely
 * output, and before another value is to be output.
 *<p>
 * Default handling (without pretty-printing) will output a single
 * comma to separate the two. Pretty-printer is
 * to output a comma as well, but can surround that with other
 * (white-space) decoration.
 */
public void writeObjectEntrySeparator(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    jg.writeRaw(',');
    _objectIndenter.writeIndentation(jg, _nesting);
}

public void writeEndObject(JsonGenerator jg, int nrOfEntries)
    throws IOException, JsonGenerationException
{
    if (!_objectIndenter.isInline()) {
        --_nesting;
    }
    if (nrOfEntries > 0) {
        _objectIndenter.writeIndentation(jg, _nesting);
    } else {
        jg.writeRaw(' ');
    }
    jg.writeRaw('}');
}

public void writeStartArray(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    if (!_arrayIndenter.isInline()) {
        ++_nesting;
    }
    jg.writeRaw('[');
}

public void beforeArrayValues(JsonGenerator jg)
    throws IOException, JsonGenerationException
{
    _arrayIndenter.writeIndentation(jg, _nesting);
}
```

```
/**
 * Method called after an array value has been completely
 * output, and before another value is to be output.
 *<p>
 * Default handling (without pretty-printing) will output a single
 * comma to separate the two. Pretty-printer is
 * to output a comma as well, but can surround that with other
 * (white-space) decoration.
 */
public void writeArrayValueSeparator(JsonGenerator jg)
   throws IOException, JsonGenerationException
{
   jg.writeRaw(',');
   _arrayIndenter.writeIndentation(jg, _nesting);
}

public void writeEndArray(JsonGenerator jg, int nrOfValues)
   throws IOException, JsonGenerationException
{
   if (!_arrayIndenter.isInline()) {
      --_nesting;
   }
   if (nrOfValues > 0) {
      _arrayIndenter.writeIndentation(jg, _nesting);
   } else {
      jg.writeRaw(' ');
   }
   jg.writeRaw(']');
}

/*
/**********************************************************
/* Helper classes
/**********************************************************
 */

/**
 * Dummy implementation that adds no indentation whatsoever
 */
public static class NopIndenter
   implements Indenter
{
   public NopIndenter() { }
   public void writeIndentation(JsonGenerator jg, int level) { }
   public boolean isInline() { return true; }
}
```

```
/**
 * This is a very simple indenter that only every adds a
 * single space for indentation. It is used as the default
 * indenter for array values.
 */
public static class FixedSpaceIndenter
   implements Indenter
{
   public FixedSpaceIndenter() { }

   public void writeIndentation(JsonGenerator jg, int level)
      throws IOException, JsonGenerationException
   {
      jg.writeRaw(' ');
   }

   public boolean isInline() { return true; }
}

/**
 * Default linefeed-based indenter uses system-specific linefeeds and
 * 2 spaces for indentation per level.
 */
public static class Lf2SpacesIndenter
   implements Indenter
{
   final static String SYSTEM_LINE_SEPARATOR;
   static {
      String lf = null;
      try {
         lf = System.getProperty("line.separator");
      } catch (Throwable t) { } // access exception?
      SYSTEM_LINE_SEPARATOR = (lf == null) ? "\n" : lf;
   }

   final static int SPACE_COUNT = 64;
   final static char[] SPACES = new char[SPACE_COUNT];
   static {
      Arrays.fill(SPACES, ' ');
   }

   public Lf2SpacesIndenter() { }

   public boolean isInline() { return false; }

   public void writeIndentation(JsonGenerator jg, int level)
      throws IOException, JsonGenerationException
   {
```

```
            jg.writeRaw(SYSTEM_LINE_SEPARATOR);
            level += level; // 2 spaces per level
            while (level > SPACE_COUNT) { // should never happen but...
               jg.writeRaw(SPACES, 0, SPACE_COUNT);
               level -= SPACES.length;
            }
            jg.writeRaw(SPACES, 0, level);
         }
      }
   }
```

```
package org.codehaus.jackson.util;

import java.util.Arrays;

public final class CharTypes
{
  private final static char[] HEX_CHARS = "0123456789ABCDEF".toCharArray();
  private final static byte[] HEX_BYTES;
  static {
     int len = HEX_CHARS.length;
     HEX_BYTES = new byte[len];
     for (int i = 0; i < len; ++i) {
        HEX_BYTES[i] = (byte) HEX_CHARS[i];
     }
  }


  /**
   * Lookup table used for determining which input characters
   * need special handling when contained in text segment.
   */
  final static int[] sInputCodes;
  static {
     /* 96 would do for most cases (backslash is ascii 94)
      * but if we want to do lookups by raw bytes it's better
      * to have full table
      */
     int[] table = new int[256];
     // Control chars and non-space white space are not allowed unquoted
     for (int i = 0; i < 32; ++i) {
        table[i] = -1;
     }
     // And then string end and quote markers are special too
     table[""] = 1;
     table['\\'] = 1;
     sInputCodes = table;
  }
```

```
/**
 * Additionally we can combine UTF-8 decoding info into similar
 * data table.
 */
final static int[] sInputCodesUtf8;
static {
    int[] table = new int[sInputCodes.length];
    System.arraycopy(sInputCodes, 0, table, 0, sInputCodes.length);
    for (int c = 128; c < 256; ++c) {
        int code;

        // We'll add number of bytes needed for decoding
        if ((c & 0xE0) == 0xC0) { // 2 bytes (0x0080 - 0x07FF)
            code = 2;
        } else if ((c & 0xF0) == 0xE0) { // 3 bytes (0x0800 - 0xFFFF)
            code = 3;
        } else if ((c & 0xF8) == 0xF0) {
            // 4 bytes; double-char with surrogates and all...
            code = 4;
        } else {
            // And -1 seems like a good "universal" error marker...
            code = -1;
        }
        table[c] = code;
    }
    sInputCodesUtf8 = table;
}

/**
 * To support non-default (and -standard) unquoted field names mode,
 * need to have alternate checking.
 * Basically this is list of 8-bit ascii characters that are legal
 * as part of Javascript identifier
 *
 * @since 1.2
 */
final static int[] sInputCodesJsNames;
static {
    int[] table = new int[256];
    // Default is "not a name char", mark ones that are
    Arrays.fill(table, -1);
    // Assume rules with JS same as Java (change if/as needed)
    for (int i = 33; i < 256; ++i) {
        if (Character.isJavaIdentifierPart((char) i)) {
            table[i] = 0;
        }
    }
    /* As per [JACKSON-267], '@', '#' and '*' are also to be accepted as well.
```

```
     * And '-' (for hyphenated names); and '+' for sake of symmetricity...
     */
    table['@'] = 0;
    table['#'] = 0;
    table['*'] = 0;
    table['-'] = 0;
    table['+'] = 0;
    sInputCodesJsNames = table;
}

/**
 * This table is similar to Latin1, except that it marks all "high-bit"
 * code as ok. They will be validated at a later point, when decoding
 * name
 */
final static int[] sInputCodesUtf8JsNames;
static {
    int[] table = new int[256];
    // start with 8-bit JS names
    System.arraycopy(sInputCodesJsNames, 0, table, 0, sInputCodesJsNames.length);
    Arrays.fill(table, 128, 128, 0);
    sInputCodesUtf8JsNames = table;
}

/**
 * Decoding table used to quickly determine characters that are
 * relevant within comment content
 */
final static int[] sInputCodesComment = new int[256];
static {
    // but first: let's start with UTF-8 multi-byte markers:
    System.arraycopy(sInputCodesUtf8, 128, sInputCodesComment, 128, 128);

    // default (0) means "ok" (skip); -1 invalid, others marked by char itself
    Arrays.fill(sInputCodesComment, 0, 32, -1); // invalid white space
    sInputCodesComment['\t'] = 0; // tab is still fine
    sInputCodesComment['\n'] = '\n'; // lf/cr need to be observed, ends cpp comment
    sInputCodesComment['\r'] = '\r';
    sInputCodesComment['*'] = '*'; // end marker for c-style comments
}

/**
 * Lookup table used for determining which output characters
 * need to be quoted.
 */
final static int[] sOutputEscapes;
static {
    int[] table = new int[256];
```

```
    // Control chars need generic escape sequence
    for (int i = 0; i < 32; ++i) {
        table[i] = -(i + 1);
    }
    /* Others (and some within that range too) have explicit shorter
     * sequences
     */
    table[""] = "";
    table['\\'] = '\\';
    // Escaping of slash is optional, so let's not add it
    table[0x08] = 'b';
    table[0x09] = 't';
    table[0x0C] = 'f';
    table[0x0A] = 'n';
    table[0x0D] = 'r';
    sOutputEscapes = table;
}

/**
 * Lookup table for the first 128 Unicode characters (7-bit ascii)
 * range. For actual hex digits, contains corresponding value;
 * for others -1.
 */
final static int[] sHexValues = new int[128];
static {
    Arrays.fill(sHexValues, -1);
    for (int i = 0; i < 10; ++i) {
        sHexValues['0' + i] = i;
    }
    for (int i = 0; i < 6; ++i) {
        sHexValues['a' + i] = 10 + i;
        sHexValues['A' + i] = 10 + i;
    }
}

public final static int[] getInputCodeLatin1() { return sInputCodes; }
public final static int[] getInputCodeUtf8() { return sInputCodesUtf8; }

public final static int[] getInputCodeLatin1JsNames() { return sInputCodesJsNames; }
public final static int[] getInputCodeUtf8JsNames() { return sInputCodesUtf8JsNames; }

public final static int[] getInputCodeComment() { return sInputCodesComment; }
public final static int[] getOutputEscapes() { return sOutputEscapes; }

public static int charToHex(int ch)
{
    return (ch > 127) ? -1 : sHexValues[ch];
}
```

```
  public static void appendQuoted(StringBuilder sb, String content)
  {
     final int[] escCodes = sOutputEscapes;
     int escLen = escCodes.length;
     for (int i = 0, len = content.length(); i < len; ++i) {
        char c = content.charAt(i);
        if (c >= escLen || escCodes[c] == 0) {
           sb.append(c);
           continue;
        }
        sb.append('\\');
        int escCode = escCodes[c];
        if (escCode < 0) { // generic quoting (hex value)
           // We know that it has to fit in just 2 hex chars
           sb.append('u');
           sb.append('0');
           sb.append('0');
           int value = -(escCode + 1);
           sb.append(HEX_CHARS[value >> 4]);
           sb.append(HEX_CHARS[value & 0xF]);
        } else { // "named", i.e. prepend with slash
           sb.append((char) escCode);
        }
     }
  }

  /**
   * @since 1.6
   */
  public static char[] copyHexChars()
  {
     return (char[]) HEX_CHARS.clone();
  }

  /**
   * @since 1.6
   */
  public static byte[] copyHexBytes()
  {
     return (byte[]) HEX_BYTES.clone();
  }
}
package org.codehaus.jackson.util;

import java.io.IOException;
import java.util.*;
```

```
import org.codehaus.jackson.*;

/**
* Helper class that can be used to sequence multiple physical
* {@link JsonParser}s to create a single logical sequence of
* tokens, as a single {@link JsonParser}.
*<p>
* Fairly simple use of {@link JsonParserDelegate}: only need
* to override {@link #nextToken} to handle transition
*
* @author tatu
* @since 1.5
*/
public class JsonParserSequence extends JsonParserDelegate
{
  /**
   * Parsers other than the first one (which is initially assigned
   * as delegate)
   */
  protected final JsonParser[] _parsers;

  /**
   * Index of the next parser in {@link #_parsers}.
   */
  protected int _nextParser;

  /*
   **********************************************************
   * Construction
   **********************************************************
   */

  protected JsonParserSequence(JsonParser[] parsers)
  {
     super(parsers[0]);
     _parsers = parsers;
     _nextParser = 1;
  }

  /**
   * Method that will construct a parser (possibly a sequence) that
   * contains all given sub-parsers.
   * All parsers given are checked to see if they are sequences: and
   * if so, they will be "flattened", that is, contained parsers are
   * directly added in a new sequence instead of adding sequences
   * within sequences. This is done to minimize delegation depth,
   * ideally only having just a single level of delegation.
   */
```

```
public static JsonParserSequence createFlattened(JsonParser first, JsonParser second)
{
   if (!(first instanceof JsonParserSequence || second instanceof JsonParserSequence)) {
      // simple:
      return new JsonParserSequence(new JsonParser[] { first, second });
   }
   ArrayList<JsonParser> p = new ArrayList<JsonParser>();
   if (first instanceof JsonParserSequence) {
      ((JsonParserSequence) first).addFlattenedActiveParsers(p);
   } else {
      p.add(first);
   }
   if (second instanceof JsonParserSequence) {
      ((JsonParserSequence) second).addFlattenedActiveParsers(p);
   } else {
      p.add(second);
   }
   return new JsonParserSequence(p.toArray(new JsonParser[p.size()]));
}

protected void addFlattenedActiveParsers(List<JsonParser> result)
{
   for (int i = _nextParser-1, len = _parsers.length; i < len; ++i) {
      JsonParser p = _parsers[i];
      if (p instanceof JsonParserSequence) {
         ((JsonParserSequence) p).addFlattenedActiveParsers(result);
      } else {
         result.add(p);
      }
   }
}

/*
 ******************************************************
 * Overridden methods, needed: cases where default
 * delegation does not work
 ******************************************************
 */

@Override
public void close() throws IOException
{
   do {
      delegate.close();
   } while (switchToNext());
}

@Override
```

```
public JsonToken nextToken() throws IOException, JsonParseException
{
   JsonToken t = delegate.nextToken();
   if (t != null) return t;
   while (switchToNext()) {
      t = delegate.nextToken();
      if (t != null) return t;
   }
   return null;
}

/*
/**********************************************************
/* Additional extended API
/**********************************************************
 */

/**
 * Method that is most useful for debugging or testing;
 * returns actual number of underlying parsers sequence
 * was constructed with (nor just ones remaining active)
 */
public int containedParsersCount() {
   return _parsers.length;
}

/*
/**********************************************************
/* Helper methods
/**********************************************************
 */

/**
 * Method that will switch active parser from the current one
 * to next parser in sequence, if there is another parser left,
 * making this the new delegate. Old delegate is returned if
 * switch succeeds.
 *
 * @return True if switch succeeded; false otherwise
 */
protected boolean switchToNext()
{
   if (_nextParser >= _parsers.length) {
      return false;
   }
   delegate = _parsers[_nextParser++];
   return true;
}
```

```
}
/**
* Utility classes used by Jackson Core functionality.
*/
package org.codehaus.jackson.util;
/* Jackson JSON-processor.
*
* Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi
*
* Licensed under the License specified in file LICENSE, included with
* the source code and binary code bundles.
* You may not use this file except in compliance with the License.
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package org.codehaus.jackson;

/**
* Shared base class for streaming processing contexts used during
* reading and writing of Json content using Streaming API.
* This context is also exposed to applications:
* context object can be used by applications to get an idea of
* relative position of the parser/generator within json content
* being processed. This allows for some contextual processing: for
* example, output within Array context can differ from that of
* Object context.
*/
public abstract class JsonStreamContext
{
  // // // Type constants used internally

  protected final static int TYPE_ROOT = 0;
  protected final static int TYPE_ARRAY = 1;
  protected final static int TYPE_OBJECT = 2;

  protected int _type;

  /**
   * Index of the currently processed entry. Starts with -1 to signal
   * that no entries have been started, and gets advanced each
   * time a new entry is started, either by encountering an expected
   * separator, or with new values if no separators are expected
   * (the case for root context).
```

```
 */
protected int _index;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

protected JsonStreamContext() { }

/*
/**********************************************************
/* Public API, accessors
/**********************************************************
 */

/**
 * Accessor for finding parent context of this context; will
 * return null for root context.
 */
public abstract JsonStreamContext getParent();

/**
 * Method that returns true if this context is an Array context;
 * that is, content is being read from or written to a Json Array.
 */
public final boolean inArray() { return _type == TYPE_ARRAY; }

/**
 * Method that returns true if this context is a Root context;
 * that is, content is being read from or written to without
 * enclosing array or object structure.
 */
public final boolean inRoot() { return _type == TYPE_ROOT; }

/**
 * Method that returns true if this context is an Object context;
 * that is, content is being read from or written to a Json Object.
 */
public final boolean inObject() { return _type == TYPE_OBJECT; }

/**
 * Method for accessing simple type description of current context;
 * either ROOT (for root-level values), OBJECT (for field names and
 * values of JSON Objects) or ARRAY (for values of JSON Arrays)
 */
public final String getTypeDesc() {
```

```
        switch (_type) {
        case TYPE_ROOT: return "ROOT";
        case TYPE_ARRAY: return "ARRAY";
        case TYPE_OBJECT: return "OBJECT";
        }
        return "?";
    }

    /**
     * @return Number of entries that are complete and started.
     */
    public final int getEntryCount()
    {
        return _index + 1;
    }

    /**
     * @return Index of the currently processed entry, if any
     */
    public final int getCurrentIndex()
    {
        return (_index < 0) ? 0 : _index;
    }

    /**
     * Method for accessing name associated with the current location.
     * Non-null for <code>FIELD_NAME</code> and value events that directly
     * follow field names; null for root level and array values.
     */
    public abstract String getCurrentName();
}
/**
* Main public API classes of the core streaming JSON
* processor: most importantly {@link org.codehaus.jackson.JsonFactory}
* used for constructing
* JSON parser ({@link org.codehaus.jackson.JsonParser})
* and generator
* ({@link org.codehaus.jackson.JsonParser})
* instances.
* <p>
* Public API of the higher-level mapping interfaces ("Mapping API")
* is found from
* under {@link org.codehaus.jackson.map} and not included here,
* except for following base interfaces:
* <ul>
*<li>{@link org.codehaus.jackson.JsonNode} is included
*within Streaming API to support integration of the Tree Model
*(which is based on <code>JsonNode</code>) with the basic
```

```
*parsers and generators (iff using mapping-supporting factory: which
*is part of Mapping API, not core)
*  </li>
*<li>{@link org.codehaus.jackson.ObjectCodec} is included so that
*  reference to the object capable of serializing/deserializing
*  Objects to/from JSON (usually, {@link org.codehaus.jackson.map.ObjectMapper})
*  can be exposed, without adding direct dependency to implementation.
*  </li>
*</ul>
* </ul>
*/

package org.codehaus.jackson;
/* Jackson JSON-processor.
*
* Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi
*
* Licensed under the License specified in file LICENSE, included with
* the source code and binary code bundles.
* You may not use this file except in compliance with the License.
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package org.codehaus.jackson;

import java.io.*;
import java.lang.ref.SoftReference;
import java.net.URL;

import org.codehaus.jackson.io.*;
import org.codehaus.jackson.impl.ByteSourceBootstrapper;
import org.codehaus.jackson.impl.ReaderBasedParser;
import org.codehaus.jackson.impl.Utf8Generator;
import org.codehaus.jackson.impl.WriterBasedGenerator;
import org.codehaus.jackson.sym.BytesToNameCanonicalizer;
import org.codehaus.jackson.sym.CharsToNameCanonicalizer;
import org.codehaus.jackson.util.BufferRecycler;
import org.codehaus.jackson.util.VersionUtil;

/**
* The main factory class of Jackson package, used to configure and
* construct reader (aka parser, {@link JsonParser})
* and writer (aka generator, {@link JsonGenerator})
* instances.
```

```
*<p>
* Factory instances are thread-safe and reusable after configuration
* (if any). Typically applications and services use only a single
* globally shared factory instance, unless they need differently
* configured factories. Factory reuse is important if efficiency matters;
* most recycling of expensive construct is done on per-factory basis.
*<p>
* Creation of a factory instance is a light-weight operation,
* and since there is no need for pluggable alternative implementations
* (as there is no "standard" JSON processor API to implement),
* the default constructor is used for constructing factory
* instances.
*
* @author Tatu Saloranta
*/
public class JsonFactory
  implements Versioned
{
  /**
   * Bitfield (set of flags) of all parser features that are enabled
   * by default.
   */
  final static int DEFAULT_PARSER_FEATURE_FLAGS = JsonParser.Feature.collectDefaults();

  /**
   * Bitfield (set of flags) of all generator features that are enabled
   * by default.
   */
  final static int DEFAULT_GENERATOR_FEATURE_FLAGS = JsonGenerator.Feature.collectDefaults();

  /*
  /**********************************************************
  /* Buffer, symbol table management
  /**********************************************************
   */

  /**
   * This <code>ThreadLocal</code> contains a {@link java.lang.ref.SoftRerefence}
   * to a {@link BufferRecycler} used to provide a low-cost
   * buffer recycling between reader and writer instances.
   */
  final protected static ThreadLocal<SoftReference<BufferRecycler>> _recyclerRef
     = new ThreadLocal<SoftReference<BufferRecycler>>();

  /**
   * Each factory comes equipped with a shared root symbol table.
   * It should not be linked back to the original blueprint, to
   * avoid contents from leaking between factories.
```

```
 */
protected CharsToNameCanonicalizer _rootCharSymbols = CharsToNameCanonicalizer.createRoot();

/**
 * Alternative to the basic symbol table, some stream-based
 * parsers use different name canonicalization method.
 *<p>
 * TODO: should clean up this; looks messy having 2 alternatives
 * with not very clear differences.
 */
protected BytesToNameCanonicalizer _rootByteSymbols = BytesToNameCanonicalizer.createRoot();

/*
/**********************************************************
/* Configuration
/**********************************************************
 */

/**
 * Object that implements conversion functionality between
 * Java objects and JSON content. For base JsonFactory implementation
 * usually not set by default, but can be explicitly set.
 * Sub-classes (like @link org.codehaus.jackson.map.MappingJsonFactory}
 * usually provide an implementation.
 */
protected ObjectCodec _objectCodec;

protected int _parserFeatures = DEFAULT_PARSER_FEATURE_FLAGS;

protected int _generatorFeatures = DEFAULT_GENERATOR_FEATURE_FLAGS;

/**
 * Default constructor used to create factory instances.
 * Creation of a factory instance is a light-weight operation,
 * but it is still a good idea to reuse limited number of
 * factory instances (and quite often just a single instance):
 * factories are used as context for storing some reused
 * processing objects (such as symbol tables parsers use)
 * and this reuse only works within context of a single
 * factory instance.
 */
public JsonFactory() { this(null); }

public JsonFactory(ObjectCodec oc) { _objectCodec = oc; }

/*
/**********************************************************
/* Versioned
```

```
/**********************************************************
 */

@Override
public Version version() {
    // VERSION is included under impl, so can't pass this class:
    return VersionUtil.versionFor(Utf8Generator.class);
}

/*
/**********************************************************
/* Configuration, parser settings
/**********************************************************
 */

/**
 * Method for enabling or disabling specified parser feature
 * (check {@link JsonParser.Feature} for list of features)
 *
 * @since 1.2
 */
public final JsonFactory configure(JsonParser.Feature f, boolean state)
{
    if (state) {
        enable(f);
    } else {
        disable(f);
    }
    return this;
}

/**
 * Method for enabling specified parser feature
 * (check {@link JsonParser.Feature} for list of features)
 *
 * @since 1.2
 */
public JsonFactory enable(JsonParser.Feature f) {
    _parserFeatures |= f.getMask();
    return this;
}

/**
 * Method for disabling specified parser features
 * (check {@link JsonParser.Feature} for list of features)
 *
 * @since 1.2
 */
```

```
public JsonFactory disable(JsonParser.Feature f) {
    _parserFeatures &= ~f.getMask();
    return this;
}

/**
 * Checked whether specified parser feature is enabled.
 *
 * @since 1.2
 */
public final boolean isEnabled(JsonParser.Feature f) {
    return (_parserFeatures & f.getMask()) != 0;
}

// // // Older deprecated (as of 1.2) methods

/**
 * @deprecated Use {@link #enable(JsonParser.Feature)} instead
 */
@SuppressWarnings("dep-ann")
public final void enableParserFeature(JsonParser.Feature f) {
    enable(f);
}

/**
 * @deprecated Use {@link #disable(JsonParser.Feature)} instead
 */
@SuppressWarnings("dep-ann")
public final void disableParserFeature(JsonParser.Feature f) {
    disable(f);
}

/**
 * @deprecated Use {@link #configure(JsonParser.Feature, boolean)} instead
 */
@SuppressWarnings("dep-ann")
public final void setParserFeature(JsonParser.Feature f, boolean state) {
    configure(f, state);
}

/**
 * @deprecated Use {@link #isEnabled(JsonParser.Feature)} instead
 */
@SuppressWarnings("dep-ann")
public final boolean isParserFeatureEnabled(JsonParser.Feature f) {
    return (_parserFeatures & f.getMask()) != 0;
}
```

```
/*
/**********************************************************
/* Configuration, generator settings
/**********************************************************
 */

/**
 * Method for enabling or disabling specified generator feature
 * (check {@link JsonGenerator.Feature} for list of features)
 *
 * @since 1.2
 */
public final JsonFactory configure(JsonGenerator.Feature f, boolean state) {
   if (state) {
      enable(f);
   } else {
      disable(f);
   }
   return this;
}


/**
 * Method for enabling specified generator features
 * (check {@link JsonGenerator.Feature} for list of features)
 *
 * @since 1.2
 */
public JsonFactory enable(JsonGenerator.Feature f) {
   _generatorFeatures |= f.getMask();
   return this;
}

/**
 * Method for disabling specified generator feature
 * (check {@link JsonGenerator.Feature} for list of features)
 *
 * @since 1.2
 */
public JsonFactory disable(JsonGenerator.Feature f) {
   _generatorFeatures &= ~f.getMask();
   return this;
}

/**
 * Check whether specified generator feature is enabled.
 *
 * @since 1.2
```

```
 */
public final boolean isEnabled(JsonGenerator.Feature f) {
   return (_generatorFeatures & f.getMask()) != 0;
}

// // // Older deprecated (as of 1.2) methods

/**
 * @deprecated Use {@link #enable(JsonGenerator.Feature)} instead
 */
@Deprecated
public final void enableGeneratorFeature(JsonGenerator.Feature f) {
   enable(f);
}

/**
 * @deprecated Use {@link #disable(JsonGenerator.Feature)} instead
 */
@Deprecated
public final void disableGeneratorFeature(JsonGenerator.Feature f) {
   disable(f);
}

/**
 * @deprecated Use {@link #configure(JsonGenerator.Feature, boolean)} instead
 */
@Deprecated
public final void setGeneratorFeature(JsonGenerator.Feature f, boolean state) {
   configure(f, state);
}

/**
 * @deprecated Use {@link #isEnabled(JsonGenerator.Feature)} instead
 */
@Deprecated
public final boolean isGeneratorFeatureEnabled(JsonGenerator.Feature f) {
   return isEnabled(f);
}

/*
/**********************************************************
/* Configuration, other
/**********************************************************
 */

public JsonFactory setCodec(ObjectCodec oc) {
   _objectCodec = oc;
   return this;
```

```
}

public ObjectCodec getCodec() { return _objectCodec; }

/*
/**********************************************************
/* Reader factories
/**********************************************************
 */

/**
 * Method for constructing json parser instance to parse
 * contents of specified file. Encoding is auto-detected
 * from contents according to json specification recommended
 * mechanism.
 *<p>
 * Underlying input stream (needed for reading contents)
 * will be <b>owned</b> (and managed, i.e. closed as need be) by
 * the parser, since caller has no access to it.
 *
 * @param f File that contains JSON content to parse
 */
public JsonParser createJsonParser(File f)
   throws IOException, JsonParseException
{
   return _createJsonParser(new FileInputStream(f), _createContext(f, true));
}

/**
 * Method for constructing json parser instance to parse
 * contents of resource reference by given URL.
 * Encoding is auto-detected
 * from contents according to json specification recommended
 * mechanism.
 *<p>
 * Underlying input stream (needed for reading contents)
 * will be <b>owned</b> (and managed, i.e. closed as need be) by
 * the parser, since caller has no access to it.
 *
 * @param url URL pointing to resource that contains JSON content to parse
 */
public JsonParser createJsonParser(URL url)
   throws IOException, JsonParseException
{
   return _createJsonParser(_optimizedStreamFromURL(url), _createContext(url, true));
}

/**
```

```
     * Method for constructing json parser instance to parse
     * the contents accessed via specified input stream.
     *<p>
     * The input stream will <b>not be owned</b> by
     * the parser, it will still be managed (i.e. closed if
     * end-of-stream is reacher, or parser close method called)
     * if (and only if) {@link org.codehaus.jackson.JsonParser.Feature#AUTO_CLOSE_SOURCE}
     * is enabled.
     *<p>
     * Note: no encoding argument is taken since it can always be
     * auto-detected as suggested by Json RFC.
     *
     * @param in InputStream to use for reading JSON content to parse
     */
    public JsonParser createJsonParser(InputStream in)
        throws IOException, JsonParseException
    {
        return _createJsonParser(in, _createContext(in, false));
    }

    /**
     * Method for constructing json parser instance to parse
     * the contents accessed via specified Reader.
     <p>
     * The read stream will <b>not be owned</b> by
     * the parser, it will still be managed (i.e. closed if
     * end-of-stream is reacher, or parser close method called)
     * if (and only if) {@link org.codehaus.jackson.JsonParser.Feature#AUTO_CLOSE_SOURCE}
     * is enabled.
     *<p>
     *
     * @param r Reader to use for reading JSON content to parse
     */
    public JsonParser createJsonParser(Reader r)
        throws IOException, JsonParseException
    {
    return _createJsonParser(r, _createContext(r, false));
    }

    public JsonParser createJsonParser(byte[] data)
        throws IOException, JsonParseException
    {
        return _createJsonParser(data, 0, data.length, _createContext(data, true));
    }

    public JsonParser createJsonParser(byte[] data, int offset, int len)
        throws IOException, JsonParseException
    {
```

```
return _createJsonParser(data, offset, len, _createContext(data, true));
  }

  public JsonParser createJsonParser(String content)
     throws IOException, JsonParseException
  {
// true -> we own the Reader (and must close); not a big deal
Reader r = new StringReader(content);
return _createJsonParser(r, _createContext(r, true));
  }

  /*
  /**********************************************************
  /* Generator factories
  /**********************************************************
   */

  /**
   * Method for constructing JSON generator for writing JSON content
   * using specified output stream.
   * Encoding to use must be specified, and needs to be one of available
   * types (as per JSON specification).
   *<p>
   * Underlying stream <b>is NOT owned</b> by the generator constructed,
   * so that generator will NOT close the output stream when
   * {@link JsonGenerator#close} is called (unless auto-closing
   * feature,
   * {@link org.codehaus.jackson.JsonGenerator.Feature#AUTO_CLOSE_TARGET}
   * is enabled).
   * Using application needs to close it explicitly if this is the case.
   *
   * @param out OutputStream to use for writing JSON content
   * @param enc Character encoding to use
   */
  public JsonGenerator createJsonGenerator(OutputStream out, JsonEncoding enc)
     throws IOException
  {
// false -> we won't manage the stream unless explicitly directed to
     IOContext ctxt = _createContext(out, false);
     ctxt.setEncoding(enc);
     if (enc == JsonEncoding.UTF8) {
        return _createUTF8JsonGenerator(out, ctxt);
     }
     // !!! TEST
     /*
     if (true) {
        ctxt.setEncoding(JsonEncoding.UTF8);
        return _createJsonGenerator(_createWriter(out, JsonEncoding.UTF8, ctxt), ctxt);
```

```
    }
    */
return _createJsonGenerator(_createWriter(out, enc, ctxt), ctxt);
 }

 /**
  * Method for constructing JSON generator for writing JSON content
  * using specified Writer.
  *<p>
  * Underlying stream <b>is NOT owned</b> by the generator constructed,
  * so that generator will NOT close the Reader when
  * {@link JsonGenerator#close} is called (unless auto-closing
  * feature,
  * {@link org.codehaus.jackson.JsonGenerator.Feature#AUTO_CLOSE_TARGET} is enabled).
  * Using application needs to close it explicitly.
  *
  * @param out Writer to use for writing JSON content
  */
 public JsonGenerator createJsonGenerator(Writer out)
    throws IOException
 {
    IOContext ctxt = _createContext(out, false);
return _createJsonGenerator(out, ctxt);
 }

 /**
  * Method for constructing json generator for writing json content
  * to specified file, overwriting contents it might have (or creating
  * it if such file does not yet exist).
  * Encoding to use must be specified, and needs to be one of available
  * types (as per JSON specification).
  *<p>
  * Underlying stream <b>is owned</b> by the generator constructed,
  * i.e. generator will handle closing of file when
  * {@link JsonGenerator#close} is called.
  *
  * @param f File to write contents to
  * @param enc Character encoding to use
  */
 public JsonGenerator createJsonGenerator(File f, JsonEncoding enc)
    throws IOException
 {
OutputStream out = new FileOutputStream(f);
// true -> yes, we have to manage the stream since we created it
    IOContext ctxt = _createContext(out, true);
    ctxt.setEncoding(enc);
    if (enc == JsonEncoding.UTF8) {
       return _createUTF8JsonGenerator(out, ctxt);
```

```
        }
    return _createJsonGenerator(_createWriter(out, enc, ctxt), ctxt);
    }

    /*
    /**********************************************************
    /* Internal factory methods, overridable
    /**********************************************************
     */

    /**
     * Overridable factory method that actually instantiates desired
     * context object.
     */
    protected IOContext _createContext(Object srcRef, boolean resourceManaged)
    {
        return new IOContext(_getBufferRecycler(), srcRef, resourceManaged);
    }

    /**
     * Overridable factory method that actually instantiates desired
     * parser.
     */
    protected JsonParser _createJsonParser(InputStream in, IOContext ctxt)
        throws IOException, JsonParseException
    {
        return new ByteSourceBootstrapper(ctxt, in).constructParser(_parserFeatures, _objectCodec,
    _rootByteSymbols, _rootCharSymbols);
    }

    /**
     * Overridable factory method that actually instantiates desired
     * parser.
     */
    protected JsonParser _createJsonParser(Reader r, IOContext ctxt)
    throws IOException, JsonParseException
    {
        return new ReaderBasedParser(ctxt, _parserFeatures, r, _objectCodec,
                _rootCharSymbols.makeChild(isEnabled(JsonParser.Feature.CANONICALIZE_FIELD_NAMES),
                    isEnabled(JsonParser.Feature.INTERN_FIELD_NAMES)));
    }

    /**
     * Overridable factory method that actually instantiates desired
     * parser.
     */
    protected JsonParser _createJsonParser(byte[] data, int offset, int len, IOContext ctxt)
        throws IOException, JsonParseException
```

```
    {
        // true -> managed (doesn't really matter; we have no stream!)
        return new ByteSourceBootstrapper(ctxt, data, offset, len).constructParser(_parserFeatures, _objectCodec,
_rootByteSymbols, _rootCharSymbols);
    }

    /**
     * Overridable factory method that actually instantiates desired
     * generator.
     */
    protected JsonGenerator _createJsonGenerator(Writer out, IOContext ctxt)
        throws IOException
    {
        return new WriterBasedGenerator(ctxt, _generatorFeatures, _objectCodec, out);
    }

    /**
     * Overridable factory method that actually instantiates desired
     * generator that is specifically to output content using specified
     * output stream and using UTF-8 encoding.
     */
    protected JsonGenerator _createUTF8JsonGenerator(OutputStream out, IOContext ctxt)
        throws IOException
    {
        return new Utf8Generator(ctxt, _generatorFeatures, _objectCodec, out);
    }

    /**
     * Method used by factory to create buffer recycler instances
     * for parsers and generators.
     *<p>
     * Note: only public to give access for <code>ObjectMapper</code>
     */
    public BufferRecycler _getBufferRecycler()
    {
        SoftReference<BufferRecycler> ref = _recyclerRef.get();
        BufferRecycler br = (ref == null) ? null : ref.get();

        if (br == null) {
            br = new BufferRecycler();
            _recyclerRef.set(new SoftReference<BufferRecycler>(br));
        }
        return br;
    }

    protected Writer _createWriter(OutputStream out, JsonEncoding enc, IOContext ctxt) throws IOException
    {
        // note: this should not get called any more (caller checks, dispatches)
```

```
        if (enc == JsonEncoding.UTF8) { // We have optimized writer for UTF-8
            return new UTF8Writer(ctxt, out);
        }
        // not optimal, but should do unless we really care about UTF-16/32 encoding speed
        return new OutputStreamWriter(out, enc.getJavaName());
    }

    /*
    /**********************************************************
    /* Internal factory methods, other
    /**********************************************************
     */

    /**
     * Helper methods used for constructing an optimal stream for
     * parsers to use, when input is to be read from an URL.
     * This helps when reading file content via URL.
     */
    protected InputStream _optimizedStreamFromURL(URL url)
        throws IOException
    {
        if ("file".equals(url.getProtocol())) {
            /* Can not do this if the path refers
             * to a network drive on windows. This fixes the problem;
             * might not be needed on all platforms (NFS?), but should not
             * matter a lot: performance penalty of extra wrapping is more
             * relevant when accessing local file system.
             */
            String host = url.getHost();
            if (host == null || host.length() == 0) {
                return new FileInputStream(url.getPath());
            }
        }
        return url.openStream();
    }
}
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.8.2
Created-By: 1.6.0_22-b04-307-10M3261 (Apple Inc.)
```

1.79 jackson-mapper-asl 1.9.13

1.79.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner

or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions

of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

```
 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
/* Jackson JSON-processor.
*
* Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi
*
* Licensed under the License specified in file LICENSE, included with
* the source code and binary code bundles.
* You may not use this file except in compliance with the License.
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
```

1.80 jackson-mapper-lgpl 1.7.4

1.80.1 Available under license :

```
/**
* Contains concrete {@link org.codehaus.jackson.JsonNode} implementations
* Jackson uses for the Tree model.
* These classes are public since concrete type will be needed
* for most operations that modify node trees. For read-only access concrete
* types are usually not needed.
*/
package org.codehaus.jackson.node;
package org.codehaus.jackson.node;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.SerializerProvider;

/**
* This singleton value class is used to contain explicit JSON null
* value.
*/
public final class NullNode
  extends ValueNode
{
  // // Just need a fly-weight singleton
```

```
  public final static NullNode instance = new NullNode();

  private NullNode() { }

  public static NullNode getInstance() { return instance; }

  @Override public JsonToken asToken() { return JsonToken.VALUE_NULL; }

  @Override
  public boolean isNull() { return true; }

  @Override
  public String getValueAsText() {
     return "null";
  }

  @Override
  public int getValueAsInt(int defaultValue) {
     return 0;
  }
  @Override
  public long getValueAsLong(long defaultValue) {
     return 0L;
  }
  @Override
  public double getValueAsDouble(double defaultValue) {
     return 0.0;
  }

  @Override
  public final void serialize(JsonGenerator jg, SerializerProvider provider)
     throws IOException, JsonProcessingException
  {
     jg.writeNull();
  }

  @Override
  public boolean equals(Object o)
  {
     return (o == this);
  }
}
package org.codehaus.jackson.node;

import java.io.IOException;

import org.codehaus.jackson.*;
```

```
import org.codehaus.jackson.map.SerializerProvider;

/**
* This concrete value class is used to contain boolean (true / false)
* values. Only two instances are ever created, to minimize memory
* usage
*/
public final class BooleanNode
  extends ValueNode
{
  // // Just need two instances...

  public final static BooleanNode TRUE = new BooleanNode();
  public final static BooleanNode FALSE = new BooleanNode();

  private BooleanNode() { }

  public static BooleanNode getTrue() { return TRUE; }
  public static BooleanNode getFalse() { return FALSE; }

  public static BooleanNode valueOf(boolean b) { return b ? TRUE : FALSE; }

  // Interesting... two choices...
  @Override public JsonToken asToken() {
     return (this == TRUE) ? JsonToken.VALUE_TRUE : JsonToken.VALUE_FALSE;
  }

  @Override
  public boolean isBoolean() { return true; }

  @Override
  public boolean getBooleanValue() {
     return (this == TRUE);
  }

  @Override
  public String getValueAsText() {
     return (this == TRUE) ? "true" : "false";
  }

  @Override
  public boolean getValueAsBoolean() {
     return (this == TRUE);
  }

  @Override
  public boolean getValueAsBoolean(boolean defaultValue) {
     return (this == TRUE);
```

```
    }

    @Override
    public int getValueAsInt(int defaultValue) {
        return (this == TRUE) ? 1 : 0;
    }
    @Override
    public long getValueAsLong(long defaultValue) {
        return (this == TRUE) ? 1L : 0L;
    }
    @Override
    public double getValueAsDouble(double defaultValue) {
        return (this == TRUE) ? 1.0 : 0.0;
    }

    @Override
    public final void serialize(JsonGenerator jg, SerializerProvider provider)
        throws IOException, JsonProcessingException
    {
        jg.writeBoolean(this == TRUE);
    }

    @Override
    public boolean equals(Object o)
    {
        /* Since there are only ever two instances in existence
         * can do identity comparison
         */
        return (o == this);
    }
}
package org.codehaus.jackson.node;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Value node that contains a wrapped POJO, to be serialized as
* a JSON constructed through data mapping (usually done by
* calling {@link org.codehaus.jackson.map.ObjectMapper}).
*/
public final class POJONode
    extends ValueNode
{
    protected final Object _value;
```

```
public POJONode(Object v) { _value = v; }

/*
/**********************************************************
/* Base class overrides
/**********************************************************
 */

@Override public JsonToken asToken() { return JsonToken.VALUE_EMBEDDED_OBJECT; }

@Override
public boolean isPojo() { return true; }

/*
/**********************************************************
/* General type coercions
/**********************************************************
 */

@Override
public String getValueAsText() {
   return (_value == null) ? "null" : _value.toString();
}

@Override
public boolean getValueAsBoolean(boolean defaultValue)
{
   if (_value != null && _value instanceof Boolean) {
      return ((Boolean) _value).booleanValue();
   }
   return defaultValue;
}

@Override
public int getValueAsInt(int defaultValue)
{
   if (_value instanceof Number) {
      return ((Number) _value).intValue();
   }
   return defaultValue;
}

@Override
public long getValueAsLong(long defaultValue)
{
   if (_value instanceof Number) {
      return ((Number) _value).longValue();
   }
```

```
    return defaultValue;
}

@Override
public double getValueAsDouble(double defaultValue)
{
    if (_value instanceof Number) {
        return ((Number) _value).doubleValue();
    }
    return defaultValue;
}

/*
/**********************************************************
/* Public API, serialization
/**********************************************************
 */

@Override
public final void serialize(JsonGenerator jg, SerializerProvider provider)
    throws IOException, JsonProcessingException
{
    if (_value == null) {
        jg.writeNull();
    } else {
        jg.writeObject(_value);
    }
}

/*
/**********************************************************
/* Extended API
/**********************************************************
 */

/**
 * Method that can be used to access the POJO this node wraps.
 */
public Object getPojo() { return _value; }

/*
/**********************************************************
/* Overridden standard methods
/**********************************************************
 */

@Override
public boolean equals(Object o)
```

```
    {
        if (o == this) return true;
        if (o == null) return false;
        if (o.getClass() != getClass()) { // final class, can do this
            return false;
        }
        POJONode other = (POJONode) o;
        if (_value == null) {
            return other._value == null;
        }
        return _value.equals(other._value);
    }

    @Override
    public int hashCode() { return _value.hashCode(); }

    @Override
    public String toString()
    {
        return String.valueOf(_value);
    }
}
package org.codehaus.jackson.node;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.NumberOutput;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Numeric node that contains simple 32-bit integer values.
*/
public final class IntNode
    extends NumericNode
{
    // // // Let's cache small set of common value

    final static int MIN_CANONICAL = -1;
    final static int MAX_CANONICAL = 10;

    private final static IntNode[] CANONICALS;
    static {
        int count = MAX_CANONICAL - MIN_CANONICAL + 1;
        CANONICALS = new IntNode[count];
        for (int i = 0; i < count; ++i) {
```

```
            CANONICALS[i] = new IntNode(MIN_CANONICAL + i);
        }
    }

    /**
     * Integer value this node contains
     */
    final int _value;

    /*
    ************************************************
    * Construction
    ************************************************
    */

    public IntNode(int v) { _value = v; }

    public static IntNode valueOf(int i) {
        if (i > MAX_CANONICAL || i < MIN_CANONICAL) return new IntNode(i);
        return CANONICALS[i - MIN_CANONICAL];
    }

    /*
    /**********************************************************
    /* BaseJsonNode extended API
    /**********************************************************
     */

    @Override public JsonToken asToken() { return JsonToken.VALUE_NUMBER_INT; }

    @Override
    public JsonParser.NumberType getNumberType() { return JsonParser.NumberType.INT; }

    /*
    /**********************************************************
    /* Overrridden JsonNode methods
    /**********************************************************
     */

    @Override
    public boolean isIntegralNumber() { return true; }

    @Override
    public boolean isInt() { return true; }

    @Override
    public Number getNumberValue() {
        return Integer.valueOf(_value);
```

```
    }

    @Override
    public int getIntValue() { return _value; }

    @Override
    public long getLongValue() { return (long) _value; }

    @Override
    public double getDoubleValue() { return (double) _value; }

    @Override
    public BigDecimal getDecimalValue() { return BigDecimal.valueOf(_value); }

    @Override
    public BigInteger getBigIntegerValue() { return BigInteger.valueOf(_value); }

    @Override
    public String getValueAsText() {
       return NumberOutput.toString(_value);
    }

    @Override
    public boolean getValueAsBoolean(boolean defaultValue) {
       return _value != 0;
    }

    @Override
    public final void serialize(JsonGenerator jg, SerializerProvider provider)
       throws IOException, JsonProcessingException
    {
       jg.writeNumber(_value);
    }

    @Override
    public boolean equals(Object o)
    {
       if (o == this) return true;
       if (o == null) return false;
       if (o.getClass() != getClass()) { // final class, can do this
          return false;
       }
       return ((IntNode) o)._value == _value;
    }

    @Override
       public int hashCode() { return _value; }
}
```

```
package org.codehaus.jackson.node;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.io.NumberInput;
import org.codehaus.jackson.util.ByteArrayBuilder;
import org.codehaus.jackson.util.CharTypes;

/**
* Value node that contains a text value.
*/
public final class TextNode
  extends ValueNode
{
  final static int INT_SPACE = ' ';

  final static TextNode EMPTY_STRING_NODE = new TextNode("");

  final String _value;

  public TextNode(String v) { _value = v; }

  /**
   * Factory method that should be used to construct instances.
   * For some common cases, can reuse canonical instances: currently
   * this is the case for empty Strings, in future possible for
   * others as well. If null is passed, will return null.
   *
   * @return Resulting {@link TextNode} object, if <b>v</b>
   *   is NOT null; null if it is.
   */
  public static TextNode valueOf(String v)
  {
     if (v == null) {
        return null;
     }
     if (v.length() == 0) {
        return EMPTY_STRING_NODE;
     }
     return new TextNode(v);
  }

  @Override public JsonToken asToken() { return JsonToken.VALUE_STRING; }

  /**
   * Yes indeed it is textual
```

```
 */
@Override
public boolean isTextual() { return true; }

@Override
public String getTextValue() {
   return _value;
}

/**
 * Method for accessing textual contents assuming they were
 * base64 encoded; if so, they are decoded and resulting binary
 * data is returned.
 */
public byte[] getBinaryValue(Base64Variant b64variant)
   throws IOException
{
   ByteArrayBuilder builder = new ByteArrayBuilder(100);
   final String str = _value;
   int ptr = 0;
   int len = str.length();

   main_loop:
   while (ptr < len) {
      // first, we'll skip preceding white space, if any
      char ch;
      do {
         ch = str.charAt(ptr++);
         if (ptr >= len) {
            break main_loop;
         }
      } while (ch <= INT_SPACE);
      int bits = b64variant.decodeBase64Char(ch);
      if (bits < 0) {
         _reportInvalidBase64(b64variant, ch, 0);
      }
      int decodedData = bits;
      // then second base64 char; can't get padding yet, nor ws
      if (ptr >= len) {
         _reportBase64EOF();
      }
      ch = str.charAt(ptr++);
      bits = b64variant.decodeBase64Char(ch);
      if (bits < 0) {
         _reportInvalidBase64(b64variant, ch, 1);
      }
      decodedData = (decodedData << 6) | bits;
      // third base64 char; can be padding, but not ws
```

```
            if (ptr >= len) {
                _reportBase64EOF();
            }
            ch = str.charAt(ptr++);
            bits = b64variant.decodeBase64Char(ch);

            // First branch: can get padding (-> 1 byte)
            if (bits < 0) {
                if (bits != Base64Variant.BASE64_VALUE_PADDING) {
                    _reportInvalidBase64(b64variant, ch, 2);
                }
                // Ok, must get padding
                if (ptr >= len) {
                    _reportBase64EOF();
                }
                ch = str.charAt(ptr++);
                if (!b64variant.usesPaddingChar(ch)) {
                    _reportInvalidBase64(b64variant, ch, 3, "expected padding character
'"+b64variant.getPaddingChar()+"'");
                }
                // Got 12 bits, only need 8, need to shift
                decodedData >>= 4;
                builder.append(decodedData);
                continue;
            }
            // Nope, 2 or 3 bytes
            decodedData = (decodedData << 6) | bits;
            // fourth and last base64 char; can be padding, but not ws
            if (ptr >= len) {
                _reportBase64EOF();
            }
            ch = str.charAt(ptr++);
            bits = b64variant.decodeBase64Char(ch);
            if (bits < 0) {
                if (bits != Base64Variant.BASE64_VALUE_PADDING) {
                    _reportInvalidBase64(b64variant, ch, 3);
                }
                decodedData >>= 2;
                builder.appendTwoBytes(decodedData);
            } else {
                // otherwise, our triple is now complete
                decodedData = (decodedData << 6) | bits;
                builder.appendThreeBytes(decodedData);
            }
        }
        return builder.toByteArray();
    }
```

```
@Override
public byte[] getBinaryValue() throws IOException
{
   return getBinaryValue(Base64Variants.getDefaultVariant());
}

/*
/**********************************************************
/* General type coercions
/**********************************************************
 */

@Override
public String getValueAsText() {
   return _value;
}

// note: neither fast nor elegant, but these work for now:

@Override
public boolean getValueAsBoolean(boolean defaultValue) {
   if (_value != null) {
      if ("true".equals(_value.trim())) {
         return true;
      }
   }
   return defaultValue;
}

@Override
public int getValueAsInt(int defaultValue) {
   return NumberInput.parseAsInt(_value, defaultValue);
}

@Override
public long getValueAsLong(long defaultValue) {
   return NumberInput.parseAsLong(_value, defaultValue);
}

@Override
public double getValueAsDouble(double defaultValue) {
   return NumberInput.parseAsDouble(_value, defaultValue);
}

/*
/**********************************************************
/* Serialization
/**********************************************************
```

```
 */

@Override
public final void serialize(JsonGenerator jg, SerializerProvider provider)
   throws IOException, JsonProcessingException
{
   if (_value == null) {
      jg.writeNull();
   } else {
      jg.writeString(_value);
   }
}

/*
/**********************************************************
/* Overridden standard methods
/**********************************************************
 */

@Override
public boolean equals(Object o)
{
   if (o == this) return true;
   if (o == null) return false;
   if (o.getClass() != getClass()) { // final class, can do this
      return false;
   }
   return ((TextNode) o)._value.equals(_value);
}

@Override
public int hashCode() { return _value.hashCode(); }

/**
 * Different from other values, Strings need quoting
 */
@Override
public String toString()
{
   int len = _value.length();
   len = len + 2 + (len >> 4);
   StringBuilder sb = new StringBuilder(len);
   appendQuoted(sb, _value);
   return sb.toString();
}

protected static void appendQuoted(StringBuilder sb, String content)
{
```

```
        sb.append("");
        CharTypes.appendQuoted(sb, content);
        sb.append("");
    }

    /*
    /**********************************************************
    /* Helper methods
    /**********************************************************
     */

    protected void _reportInvalidBase64(Base64Variant b64variant, char ch, int bindex)
        throws JsonParseException
    {
        _reportInvalidBase64(b64variant, ch, bindex, null);
    }

    /**
     * @param bindex Relative index within base64 character unit; between 0
     *   and 3 (as unit has exactly 4 characters)
     */
    protected void _reportInvalidBase64(Base64Variant b64variant, char ch, int bindex, String msg)
        throws JsonParseException
    {
        String base;
        if (ch <= INT_SPACE) {
            base = "Illegal white space character (code 0x"+Integer.toHexString(ch)+") as character #"+(bindex+1)+" of
4-char base64 unit: can only used between units";
        } else if (b64variant.usesPaddingChar(ch)) {
            base = "Unexpected padding character ('"+b64variant.getPaddingChar()+"') as character #"+(bindex+1)+" of
4-char base64 unit: padding only legal as 3rd or 4th character";
        } else if (!Character.isDefined(ch) || Character.isISOControl(ch)) {
            // Not sure if we can really get here... ? (most illegal xml chars are caught at lower level)
            base = "Illegal character (code 0x"+Integer.toHexString(ch)+") in base64 content";
        } else {
            base = "Illegal character '"+ch+"' (code 0x"+Integer.toHexString(ch)+") in base64 content";
        }
        if (msg != null) {
            base = base + ": " + msg;
        }
        throw new JsonParseException(base, JsonLocation.NA);
    }

    protected void _reportBase64EOF()
        throws JsonParseException
    {
        throw new JsonParseException("Unexpected end-of-String when base64 content", JsonLocation.NA);
    }
```

```
}
package org.codehaus.jackson.node;

import java.math.BigDecimal;
import java.math.BigInteger;

/**
* Base class that specifies methods for getting access to
* Node instances (newly constructed, or shared, depending
* on type), as well as basic implementation of the methods.
* Designed to be sub-classed if extended functionality (additions
* to behavior of node types, mostly) is needed.
*/
public class JsonNodeFactory
{
  /**
   * Default singleton instance that construct "standard" node instances:
   * given that this class is stateless, a globally shared singleton
   * can be used.
   */
  public final static JsonNodeFactory instance = new JsonNodeFactory();

  protected JsonNodeFactory() { }

  /*
  /**********************************************************
  /* Factory methods for literal values
  /**********************************************************
   */

  /**
   * Factory method for getting an instance of Json boolean value
   * (either literal 'true' or 'false')
   */
  public BooleanNode booleanNode(boolean v) {
     return v ? BooleanNode.getTrue() : BooleanNode.getFalse();
  }

  /**
   * Factory method for getting an instance of Json null node (which
   * represents literal null value)
   */
  public NullNode nullNode() { return NullNode.getInstance(); }

  /*
  /**********************************************************
  /* Factory methods for numeric values
  /**********************************************************
```

```
 */

/**
 * Factory method for getting an instance of Json numeric value
 * that expresses given 8-bit value
 */
public NumericNode numberNode(byte v) { return IntNode.valueOf(v); }

/**
 * Factory method for getting an instance of Json numeric value
 * that expresses given 16-bit integer value
 */
public NumericNode numberNode(short v) { return IntNode.valueOf(v); }

/**
 * Factory method for getting an instance of Json numeric value
 * that expresses given 32-bit integer value
 */
public NumericNode numberNode(int v) { return IntNode.valueOf(v); }

/**
 * Factory method for getting an instance of Json numeric value
 * that expresses given 64-bit integer value
 */
public NumericNode numberNode(long v) { return LongNode.valueOf(v); }

/**
 * Factory method for getting an instance of Json numeric value
 * that expresses given unlimited range integer value
 */
public NumericNode numberNode(BigInteger v) { return BigIntegerNode.valueOf(v); }

/**
 * Factory method for getting an instance of Json numeric value
 * that expresses given 32-bit floating point value
 */
public NumericNode numberNode(float v) { return DoubleNode.valueOf((double) v); }

/**
 * Factory method for getting an instance of Json numeric value
 * that expresses given 64-bit floating point value
 */
public NumericNode numberNode(double v) { return DoubleNode.valueOf(v); }

/**
 * Factory method for getting an instance of Json numeric value
 * that expresses given unlimited precision floating point value
 */
```

```
public NumericNode numberNode(BigDecimal v) { return DecimalNode.valueOf(v); }

/*
/**********************************************************
/* Factory methods for textual values
/**********************************************************
 */

/**
 * Factory method for constructing a node that represents Json
 * String value
 */
public TextNode textNode(String text) { return TextNode.valueOf(text); }

/**
 * Factory method for constructing a node that represents given
 * binary data, and will get serialized as equivalent base64-encoded
 * String value
 */
public BinaryNode binaryNode(byte[] data) { return BinaryNode.valueOf(data); }

/**
 * Factory method for constructing a node that represents given
 * binary data, and will get serialized as equivalent base64-encoded
 * String value
 */
public BinaryNode binaryNode(byte[] data, int offset, int length) {
   return BinaryNode.valueOf(data, offset, length);
}

/*
/**********************************************************
/* Factory method for structured values
/**********************************************************
 */

/**
 * Factory method for constructing an empty JSON Array node
 */
public ArrayNode arrayNode() { return new ArrayNode(this); }

/**
 * Factory method for constructing an empty JSON Object ("struct") node
 */
public ObjectNode objectNode() { return new ObjectNode(this); }

/**
 * Factory method for constructing a wrapper for POJO
```

```
   * ("Plain Old Java Object") objects; these will get serialized
   * using data binding, usually as JSON Objects, but in some
   * cases as JSON Strings or other node types.
   */
  public POJONode POJONode(Object pojo) { return new POJONode(pojo); }
}
package org.codehaus.jackson.node;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Numeric node that contains values that do not fit in simple
* integer (int, long) or floating point (double) values.
*/
public final class DecimalNode
  extends NumericNode
{
  final protected BigDecimal _value;

  /*
  /**********************************************************
  /* Construction
  /**********************************************************
   */

  public DecimalNode(BigDecimal v) { _value = v; }

  public static DecimalNode valueOf(BigDecimal d) { return new DecimalNode(d); }

  /*
  /**********************************************************
  /* BaseJsonNode extended API
  /**********************************************************
   */

  @Override public JsonToken asToken() { return JsonToken.VALUE_NUMBER_FLOAT; }

  @Override
  public JsonParser.NumberType getNumberType() { return JsonParser.NumberType.BIG_DECIMAL; }

  /*
  /**********************************************************
  /* Overrridden JsonNode methods
```

```
/**********************************************************
 */

@Override
public boolean isFloatingPointNumber() { return true; }

@Override
public boolean isBigDecimal() { return true; }

@Override
public Number getNumberValue() { return _value; }

@Override
public int getIntValue() { return _value.intValue(); }

@Override
public long getLongValue() { return _value.longValue(); }


@Override
public BigInteger getBigIntegerValue() { return _value.toBigInteger(); }

@Override
public double getDoubleValue() { return _value.doubleValue(); }

@Override
public BigDecimal getDecimalValue() { return _value; }

@Override
public String getValueAsText() {
   return _value.toString();
}

@Override
public final void serialize(JsonGenerator jg, SerializerProvider provider)
   throws IOException, JsonProcessingException
{
   jg.writeNumber(_value);
}

@Override
public boolean equals(Object o)
{
   if (o == this) return true;
   if (o == null) return false;
   if (o.getClass() != getClass()) { // final class, can do this
      return false;
   }
```

```
        return ((DecimalNode) o)._value.equals(_value);
    }

    @Override
    public int hashCode() { return _value.hashCode(); }
}
package org.codehaus.jackson.node;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.NumberOutput;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Numeric node that contains 64-bit ("double precision")
* floating point values simple 32-bit integer values.
*/
public final class DoubleNode
    extends NumericNode
{
    protected final double _value;

    /*
    /**********************************************************
    /* Construction
    /**********************************************************
     */

    public DoubleNode(double v) { _value = v; }

    public static DoubleNode valueOf(double v) { return new DoubleNode(v); }

    /*
    /**********************************************************
    /* BaseJsonNode extended API
    /**********************************************************
     */

    @Override public JsonToken asToken() { return JsonToken.VALUE_NUMBER_FLOAT; }

    @Override
    public JsonParser.NumberType getNumberType() { return JsonParser.NumberType.DOUBLE; }

    /*
    /**********************************************************
```

```
/* Overrridden JsonNode methods
/**********************************************************
 */

@Override
public boolean isFloatingPointNumber() { return true; }

@Override
public boolean isDouble() { return true; }

@Override
public Number getNumberValue() {
   return Double.valueOf(_value);
}

@Override
   public int getIntValue() { return (int) _value; }

@Override
   public long getLongValue() { return (long) _value; }

@Override
   public double getDoubleValue() { return _value; }

@Override
   public BigDecimal getDecimalValue() { return BigDecimal.valueOf(_value); }

@Override
public BigInteger getBigIntegerValue() {
   return getDecimalValue().toBigInteger();
}

@Override
public String getValueAsText() {
   return NumberOutput.toString(_value);
}

@Override
public final void serialize(JsonGenerator jg, SerializerProvider provider)
   throws IOException, JsonProcessingException
{
   jg.writeNumber(_value);
}

@Override
public boolean equals(Object o)
{
   if (o == this) return true;
```

```
        if (o == null) return false;
        if (o.getClass() != getClass()) { // final class, can do this
            return false;
        }
        return ((DoubleNode) o)._value == _value;
    }

    @Override
    public int hashCode()
    {
        // same as hashCode Double.class uses
        long l = Double.doubleToLongBits(_value);
        return ((int) l) ^ (int) (l >> 32);

    }
}
package org.codehaus.jackson.node;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.JsonToken;

/**
* This intermediate base class is used for all leaf nodes, that is,
* all non-container (array or object) nodes, except for the
* "missing node".
*/
public abstract class ValueNode
    extends BaseJsonNode
{
    protected ValueNode() { }

    @Override
    public boolean isValueNode() { return true; }

    @Override
    public abstract JsonToken asToken();

    /*
    ////////////////////////////////////////////////////
    // Public API, path handling
    ////////////////////////////////////////////////////
     */

    @Override
    public JsonNode path(String fieldName) { return MissingNode.getInstance(); }

    @Override
    public JsonNode path(int index) { return MissingNode.getInstance(); }
```

```
  /*
  ////////////////////////////////////////////////////
  // Base impls for standard methods
  ////////////////////////////////////////////////////
   */

  @Override
  public String toString() { return getValueAsText(); }
}
package org.codehaus.jackson.node;

import java.io.IOException;
import java.util.List;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.JsonSerializableWithType;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.TypeSerializer;

/**
* Abstract base class common to all standard {@link JsonNode}
* implementations.
* The main addition here is that we declare that sub-classes must
* implement {@link JsonSerializableWithType}.
* This simplifies object mapping
* aspects a bit, as no external serializers are needed.
*/
public abstract class BaseJsonNode
  extends JsonNode
  implements JsonSerializableWithType
{
  protected BaseJsonNode() { }

  /*
  /**********************************************************
  /* Basic definitions for non-container types
  /**********************************************************
   */

  @Override
  public JsonNode findValue(String fieldName) {
     return null;
  }

  @Override
  public final JsonNode findPath(String fieldName)
  {
```

```
    JsonNode value = findValue(fieldName);
    if (value == null) {
        return MissingNode.getInstance();
    }
    return value;
}

// note: co-variant return type
@Override
public ObjectNode findParent(String fieldName) {
    return null;
}

@Override
public List<JsonNode> findValues(String fieldName, List<JsonNode> foundSoFar) {
    return foundSoFar;
}

@Override
public List<String> findValuesAsText(String fieldName, List<String> foundSoFar) {
    return foundSoFar;
}

@Override
public List<JsonNode> findParents(String fieldName, List<JsonNode> foundSoFar) {
    return foundSoFar;
}

/*
/**********************************************************
/* Support for traversal-as-stream
/**********************************************************
 */

@Override
public JsonParser traverse() {
    return new TreeTraversingParser(this);
}

/**
 * Method that can be used for efficient type detection
 * when using stream abstraction for traversing nodes.
 * Will return the first {@link JsonToken} that equivalent
 * stream event would produce (for most nodes there is just
 * one token but for structured/container types multiple)
 *
 * @since 1.3
 */
```

```
@Override
public abstract JsonToken asToken();

/**
 * @since 1.3
 */
@Override
public JsonParser.NumberType getNumberType() {
   // most types non-numeric, so:
   return null;
}

/*
/**********************************************************
/* JsonSerializable
/**********************************************************
 */

/**
 * Method called to serialize node instances using given generator.
 */
public abstract void serialize(JsonGenerator jgen, SerializerProvider provider)
   throws IOException, JsonProcessingException;

/**
 * Since JSON node typing is only based on JSON values,
 * there is no need to include type information. So, serialize
 * the same way as when no typing is enabled.
 */
public void serializeWithType(JsonGenerator jgen, SerializerProvider provider,
      TypeSerializer typeSer)
   throws IOException, JsonProcessingException
{
   serialize(jgen, provider);
}

/*
/**********************************************************
/* Other
/**********************************************************
 */

/**
 *<p>
 * Note: this method is deprecated, given that we
 * want to use the standard serialization interface.
 */
@Override
```

```
  public final void writeTo(JsonGenerator jgen)
     throws IOException, JsonGenerationException
  {
     /* it's ok to pass null, as long as other nodes handle
      * it properly...
      */
     serialize(jgen, null);
  }
}
package org.codehaus.jackson.node;

import java.math.BigDecimal;
import java.util.Iterator;
import java.util.List;
import java.util.NoSuchElementException;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.JsonToken;

/**
* This intermediate base class is used for all container nodes,
* specifically, array and object nodes.
*/
public abstract class ContainerNode
  extends BaseJsonNode
{
  /**
   * We will keep a reference to the Object (usually TreeMapper)
   * that can construct instances of nodes to add to this container
   * node.
   */
  JsonNodeFactory _nodeFactory;

  protected ContainerNode(JsonNodeFactory nc)
  {
     _nodeFactory = nc;
  }

  @Override
  public boolean isContainerNode() { return true; }

  @Override
  public abstract JsonToken asToken();

  @Override
  public String getValueAsText() { return null; }

  /*
```

```
/**********************************************************
/* Find methods; made abstract again to ensure implementation
/**********************************************************
 */

@Override
public abstract JsonNode findValue(String fieldName);

@Override
public abstract ObjectNode findParent(String fieldName);

@Override
public abstract List<JsonNode> findValues(String fieldName, List<JsonNode> foundSoFar);

@Override
public abstract List<JsonNode> findParents(String fieldName, List<JsonNode> foundSoFar);

@Override
public abstract List<String> findValuesAsText(String fieldName, List<String> foundSoFar);

/*
/**********************************************************
/* Methods reset as abstract to force real implementation
/**********************************************************
 */

@Override
public abstract int size();

@Override
public abstract JsonNode get(int index);

@Override
public abstract JsonNode get(String fieldName);

/*
/**********************************************************
/* NodeCreator implementation, just dispatch to
/* the real creator
/**********************************************************
 */

public final ArrayNode arrayNode() { return _nodeFactory.arrayNode(); }
public final ObjectNode objectNode() { return _nodeFactory.objectNode(); }
public final NullNode nullNode() { return _nodeFactory.nullNode(); }

public final BooleanNode booleanNode(boolean v) { return _nodeFactory.booleanNode(v); }
```

```
    public final NumericNode numberNode(byte v) { return _nodeFactory.numberNode(v); }
    public final NumericNode numberNode(short v) { return _nodeFactory.numberNode(v); }
    public final NumericNode numberNode(int v) { return _nodeFactory.numberNode(v); }
    public final NumericNode numberNode(long v) { return _nodeFactory.numberNode(v); }
    public final NumericNode numberNode(float v) { return _nodeFactory.numberNode(v); }
    public final NumericNode numberNode(double v) { return _nodeFactory.numberNode(v); }
    public final NumericNode numberNode(BigDecimal v) { return (_nodeFactory.numberNode(v)); }

    public final TextNode textNode(String text) { return _nodeFactory.textNode(text); }

    public final BinaryNode binaryNode(byte[] data) { return _nodeFactory.binaryNode(data); }
    public final BinaryNode binaryNode(byte[] data, int offset, int length) { return _nodeFactory.binaryNode(data,
offset, length); }

    public final POJONode POJONode(Object pojo) { return _nodeFactory.POJONode(pojo); }

    /*
    /**********************************************************
    /* Common mutators
    /**********************************************************
     */

    /**
     * Method for removing all children container has (if any)
     *
     * @return Container node itself (to allow method call chaining)
     *
     * @since 1.3
     */
    public abstract ContainerNode removeAll();

    /*
    /**********************************************************
    /* Helper classes
    /**********************************************************
     */

    protected static class NoNodesIterator
        implements Iterator<JsonNode>
    {
        final static NoNodesIterator instance = new NoNodesIterator();

        private NoNodesIterator() { }

        public static NoNodesIterator instance() { return instance; }

        public boolean hasNext() { return false; }
        public JsonNode next() { throw new NoSuchElementException(); }
```

```
        public void remove() {
            // could as well throw IllegalOperationException?
            throw new IllegalStateException();
        }
    }

    protected static class NoStringsIterator
        implements Iterator<String>
    {
        final static NoStringsIterator instance = new NoStringsIterator();

        private NoStringsIterator() { }

        public static NoStringsIterator instance() { return instance; }

        public boolean hasNext() { return false; }
        public String next() { throw new NoSuchElementException(); }

        public void remove() {
            // could as well throw IllegalOperationException?
            throw new IllegalStateException();
        }
    }
}
package org.codehaus.jackson.node;

import java.io.IOException;
import java.math.BigDecimal;
import java.util.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Node class that represents Arrays mapped from Json content.
*/
public final class ArrayNode
    extends ContainerNode
{
    protected ArrayList<JsonNode> _children;

    public ArrayNode(JsonNodeFactory nc) { super(nc); }

    /*
    /**********************************************************
    /* Implementation of core JsonNode API
    /**********************************************************
```

```
 */

@Override public JsonToken asToken() { return JsonToken.START_ARRAY; }

@Override
public boolean isArray() { return true; }

@Override
public int size()
{
   return (_children == null) ? 0 : _children.size();
}

@Override
public Iterator<JsonNode> getElements()
{
   return (_children == null) ? NoNodesIterator.instance() : _children.iterator();
}

@Override
public JsonNode get(int index)
{
   if (index >= 0 && (_children != null) && index < _children.size()) {
      return _children.get(index);
   }
   return null;
}

@Override
public JsonNode get(String fieldName) { return null; }

@Override
public JsonNode path(String fieldName) { return MissingNode.getInstance(); }

@Override
public JsonNode path(int index)
{
   if (index >= 0 && (_children != null) && index < _children.size()) {
      return _children.get(index);
   }
   return MissingNode.getInstance();
}

/*
/**********************************************************
/* Public API, serialization
/**********************************************************
 */
```

```
@Override
public final void serialize(JsonGenerator jg, SerializerProvider provider)
   throws IOException, JsonProcessingException
{
   jg.writeStartArray();
   if (_children != null) {
      for (JsonNode n : _children) {
         /* 17-Feb-2009, tatu: Can we trust that all nodes will always
          *   extend BaseJsonNode? Or if not, at least implement
          *   JsonSerializable? Let's start with former, change if
          *   we must.
          */
         ((BaseJsonNode)n).writeTo(jg);
      }
   }
   jg.writeEndArray();
}

/*
/**********************************************************
/* Public API, finding value nodes
/**********************************************************
 */

@Override
public JsonNode findValue(String fieldName)
{
   if (_children != null) {
      for (JsonNode node : _children) {
         JsonNode value = node.findValue(fieldName);
         if (value != null) {
            return value;
         }
      }
   }
   return null;
}

@Override
public List<JsonNode> findValues(String fieldName, List<JsonNode> foundSoFar)
{
   if (_children != null) {
      for (JsonNode node : _children) {
         foundSoFar = node.findValues(fieldName, foundSoFar);
      }
   }
   return foundSoFar;
```

```
}

@Override
public List<String> findValuesAsText(String fieldName, List<String> foundSoFar)
{
   if (_children != null) {
      for (JsonNode node : _children) {
         foundSoFar = node.findValuesAsText(fieldName, foundSoFar);
      }
   }
   return foundSoFar;
}

@Override
public ObjectNode findParent(String fieldName)
{
   if (_children != null) {
      for (JsonNode node : _children) {
         JsonNode parent = node.findParent(fieldName);
         if (parent != null) {
            return (ObjectNode) parent;
         }
      }
   }
   return null;
}

@Override
public List<JsonNode> findParents(String fieldName, List<JsonNode> foundSoFar)
{
   if (_children != null) {
      for (JsonNode node : _children) {
         foundSoFar = node.findParents(fieldName, foundSoFar);
      }
   }
   return foundSoFar;
}

/*
/**********************************************************
/* Extended ObjectNode API, accessors
/**********************************************************
 */

/**
 * Method that will set specified field, replacing old value,
 * if any.
 *
```

```
     * @param value to set field to; if null, will be converted
     *   to a {@link NullNode} first  (to remove field entry, call
     *   {@link #remove} instead)
     *
     * @return Old value of the field, if any; null if there was no
     *   old value.
     */
    public JsonNode set(int index, JsonNode value)
    {
        if (value == null) { // let's not store 'raw' nulls but nodes
            value = nullNode();
        }
        return _set(index, value);
    }

    public void add(JsonNode value)
    {
        if (value == null) { // let's not store 'raw' nulls but nodes
            value = nullNode();
        }
        _add(value);
    }

    /**
     * Method for adding all child nodes of given Array, appending to
     * child nodes this array contains
     *
     * @param other Array to add contents from
     *
     * @return This node (to allow chaining)
     *
     * @since 1.3
     */
    public JsonNode addAll(ArrayNode other)
    {
        int len = other.size();
        if (len > 0) {
            if (_children == null) {
                _children = new ArrayList<JsonNode>(len+2);
            }
            other.addContentsTo(_children);
        }
        return this;
    }

    /**
     * Method for adding given nodes as child nodes of this array node.
     *
```

```
     * @param nodes Nodes to add
     *
     * @return This node (to allow chaining)
     *
     * @since 1.3
     */
    public JsonNode addAll(Collection<JsonNode> nodes)
    {
        int len = nodes.size();
        if (len > 0) {
            if (_children == null) {
                _children = new ArrayList<JsonNode>(nodes);
            } else {
                _children.addAll(nodes);
            }
        }
        return this;
    }

    /**
     * Method for inserting specified child node as an element
     * of this Array. If index is 0 or less, it will be inserted as
     * the first element; if >= size(), appended at the end, and otherwise
     * inserted before existing element in specified index.
     * No exceptions are thrown for any index.
     */
    public void insert(int index, JsonNode value)
    {
        if (value == null) {
            value = nullNode();
        }
        _insert(index, value);
    }

    /**
     * Method for removing an entry from this ArrayNode.
     * Will return value of the entry at specified index, if entry existed;
     * null if not.
     */
    public JsonNode remove(int index)
    {
        if (index >= 0 && (_children != null) && index < _children.size()) {
            return _children.remove(index);
        }
        return null;
    }

    @Override
```

```
public ArrayNode removeAll()
{
   _children = null;
   return this;
}

/*
/**********************************************************
/* Extended ObjectNode API, mutators, generic
/**********************************************************
 */

/**
 * Method that will construct an ArrayNode and add it as a
 * field of this ObjectNode, replacing old value, if any.
 *
 * @return Newly constructed ArrayNode
 */
public ArrayNode addArray()
{
   ArrayNode n  = arrayNode();
   _add(n);
   return n;
}

/**
 * Method that will construct an ObjectNode and add it at the end
 * of this array node.
 *
 * @return Newly constructed ObjectNode
 */
public ObjectNode addObject()
{
   ObjectNode n  = objectNode();
   _add(n);
   return n;
}

/**
 * Method that will construct a POJONode and add it at the end
 * of this array node.
 */
public void addPOJO(Object value)
{
   if (value == null) {
      addNull();
   } else {
      _add(POJONode(value));
```

```
    }
}

public void addNull()
{
    _add(nullNode());
}

/**
 * Method for setting value of a field to specified numeric value.
 */
public void add(int v) { _add(numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void add(long v) { _add(numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void add(float v) { _add(numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void add(double v) { _add(numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void add(BigDecimal v) {
    if (v == null) {
        addNull();
    } else {
        _add(numberNode(v));
    }
}

/**
 * Method for setting value of a field to specified String value.
 */
public void add(String v) {
    if (v == null) {
        addNull();
    } else {
        _add(textNode(v));
    }
```

```
}

/**
 * Method for setting value of a field to specified String value.
 */
public void add(boolean v) { _add(booleanNode(v)); }

/**
 * Method for setting value of a field to specified binary value
 */
public void add(byte[] v) {
    if (v == null) {
        addNull();
    } else {
        _add(binaryNode(v));
    }
}

public ArrayNode insertArray(int index)
{
    ArrayNode n  = arrayNode();
    _insert(index, n);
    return n;
}

/**
 * Method that will construct an ObjectNode and add it at the end
 * of this array node.
 *
 * @return Newly constructed ObjectNode
 */
public ObjectNode insertObject(int index)
{
    ObjectNode n  = objectNode();
    _insert(index, n);
    return n;
}

/**
 * Method that will construct a POJONode and add it at the end
 * of this array node.
 */
public void insertPOJO(int index, Object value)
{
    if (value == null) {
        insertNull(index);
    } else {
        _insert(index, POJONode(value));
```

```
    }
}

public void insertNull(int index)
{
    _insert(index, nullNode());
}

/**
 * Method for setting value of a field to specified numeric value.
 */
public void insert(int index, int v) { _insert(index, numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void insert(int index, long v) { _insert(index, numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void insert(int index, float v) { _insert(index, numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void insert(int index, double v) { _insert(index, numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void insert(int index, BigDecimal v) {
    if (v == null) {
        insertNull(index);
    } else {
        _insert(index, numberNode(v));
    }
}

/**
 * Method for setting value of a field to specified String value.
 */
public void insert(int index, String v) {
    if (v == null) {
        insertNull(index);
    } else {
        _insert(index, textNode(v));
    }
```

```
}

/**
 * Method for setting value of a field to specified String value.
 */
public void insert(int index, boolean v) { _insert(index, booleanNode(v)); }

/**
 * Method for setting value of a field to specified binary value
 */
public void insert(int index, byte[] v) {
    if (v == null) {
        insertNull(index);
    } else {
        _insert(index, binaryNode(v));
    }
}

/*
/**********************************************************
/* Package methods (for other node classes to use)
/**********************************************************
 */

/**
 * @since 1.6
 */
protected void addContentsTo(List<JsonNode> dst)
{
    if (_children != null) {
        for (JsonNode n : _children) {
            dst.add(n);
        }
    }
}

/*
/**********************************************************
/* Standard methods
/**********************************************************
 */

@Override
public boolean equals(Object o)
{
    if (o == this) return true;
    if (o == null) return false;
    if (o.getClass() != getClass()) { // final class, can do this
```

```
            return false;
        }
        ArrayNode other = (ArrayNode) o;
        if (_children == null || _children.size() == 0) {
            return other.size() == 0;
        }
        return other._sameChildren(_children);
    }

    @Override
    public int hashCode()
    {
        int hash;
        if (_children == null) {
            hash = 1;
        } else {
            hash = _children.size();
            for (JsonNode n : _children) {
                if (n != null) {
                    hash ^= n.hashCode();
                }
            }
        }
        return hash;
    }


    @Override
    public String toString()
    {
        StringBuilder sb = new StringBuilder(16 + (size() << 4));
        sb.append('[');
        if (_children != null) {
            for (int i = 0, len = _children.size(); i < len; ++i) {
                if (i > 0) {
                    sb.append(',');
                }
                sb.append(_children.get(i).toString());
            }
        }
        sb.append(']');
        return sb.toString();
    }

    /*
    /**********************************************************
    /* Internal methods
    /**********************************************************
```

```
 */

public JsonNode _set(int index, JsonNode value)
{
   if (_children == null || index < 0 || index >= _children.size()) {
      throw new IndexOutOfBoundsException("Illegal index "+index+", array size "+size());
   }
   return _children.set(index, value);
}

private void _add(JsonNode node)
{
   if (_children == null) {
      _children = new ArrayList<JsonNode>();
   }
   _children.add(node);
}

private void _insert(int index, JsonNode node)
{
   if (_children == null) {
      _children = new ArrayList<JsonNode>();
      _children.add(node);
      return;
   }
   if (index < 0) {
      _children.add(0, node);
   } else if (index >= _children.size()) {
      _children.add(node);
   } else {
      _children.add(index, node);
   }
}

/**
 * Note: this method gets called iff <code>otherChildren</code>
 * is non-empty
 */
private boolean _sameChildren(ArrayList<JsonNode> otherChildren)
{
   int len = otherChildren.size();
   if (this.size() != len) { // important: call size() to handle case of null list...
      return false;
   }
   for (int i = 0; i < len; ++i) {
      if (!_children.get(i).equals(otherChildren.get(i))) {
         return false;
      }
```

```
      }
      return true;
    }
  }
package org.codehaus.jackson.node;

import java.util.*;

import org.codehaus.jackson.*;

/**
* Helper class used by {@link TreeTraversingParser} to keep track
* of current location within traversed JSON tree.
*/
abstract class NodeCursor
  extends JsonStreamContext
{
  /**
   * Parent cursor of this cursor, if any; null for root
   * cursors.
   */
  final NodeCursor _parent;

  public NodeCursor(int contextType, NodeCursor p)
  {
    super();
    _type = contextType;
    _index = -1;
    _parent = p;
  }

  /*
  /**********************************************************
  /* JsonStreamContext impl
  /**********************************************************
   */

  // note: co-variant return type
  @Override
  public final NodeCursor getParent() { return _parent; }

  @Override
  public abstract String getCurrentName();

  /*
  /**********************************************************
  /* Extended API
  /**********************************************************
```

```
 */

public abstract JsonToken nextToken();
public abstract JsonToken nextValue();
public abstract JsonToken endToken();

public abstract JsonNode currentNode();
public abstract boolean currentHasChildren();

/**
 * Method called to create a new context for iterating all
 * contents of the current structured value (JSON array or object)
 */
public final NodeCursor iterateChildren() {
   JsonNode n = currentNode();
   if (n == null) throw new IllegalStateException("No current node");
   if (n.isArray()) { // false since we have already returned START_ARRAY
      return new Array(n, this);
   }
   if (n.isObject()) {
      return new Object(n, this);
   }
   throw new IllegalStateException("Current node of type "+n.getClass().getName());
}

/*
/**********************************************************
/* Concrete implementations
/**********************************************************
 */

/**
 * Context matching root-level value nodes (i.e. anything other
 * than JSON Object and Array).
 * Note that context is NOT created for leaf values.
 */
protected final static class RootValue
   extends NodeCursor
{
   JsonNode _node;

   protected boolean _done = false;

   public RootValue(JsonNode n, NodeCursor p) {
      super(JsonStreamContext.TYPE_ROOT, p);
      _node = n;
   }
```

```
    @Override
    public String getCurrentName() { return null; }

    @Override
    public JsonToken nextToken() {
        if (!_done) {
            _done = true;
            return _node.asToken();
        }
        _node = null;
        return null;
    }

    @Override
    public JsonToken nextValue() { return nextToken(); }
    @Override
    public JsonToken endToken() { return null; }
    @Override
    public JsonNode currentNode() { return _node; }
    @Override
    public boolean currentHasChildren() { return false; }
}

/**
 * Cursor used for traversing non-empty JSON Array nodes
 */
protected final static class Array
    extends NodeCursor
{
    Iterator<JsonNode> _contents;

    JsonNode _currentNode;

    public Array(JsonNode n, NodeCursor p) {
        super(JsonStreamContext.TYPE_ARRAY, p);
        _contents = n.getElements();
    }

    @Override
    public String getCurrentName() { return null; }

    @Override
    public JsonToken nextToken()
    {
        if (!_contents.hasNext()) {
            _currentNode = null;
            return null;
        }
```

```
        _currentNode = _contents.next();
        return _currentNode.asToken();
    }

    @Override
    public JsonToken nextValue() { return nextToken(); }
    @Override
    public JsonToken endToken() { return JsonToken.END_ARRAY; }

    @Override
    public JsonNode currentNode() { return _currentNode; }
    @Override
    public boolean currentHasChildren() {
        // note: ONLY to be called for container nodes
        return ((ContainerNode) currentNode()).size() > 0;
    }
}

/**
 * Cursor used for traversing non-empty JSON Object nodes
 */
protected final static class Object
    extends NodeCursor
{
    Iterator<Map.Entry<String, JsonNode>> _contents;
    Map.Entry<String, JsonNode> _current;

    boolean _needEntry;

    public Object(JsonNode n, NodeCursor p)
    {
        super(JsonStreamContext.TYPE_OBJECT, p);
        _contents = ((ObjectNode) n).getFields();
        _needEntry = true;
    }

    @Override
    public String getCurrentName() {
        return (_current == null) ? null : _current.getKey();
    }

    @Override
    public JsonToken nextToken()
    {
        // Need a new entry?
        if (_needEntry) {
            if (!_contents.hasNext()) {
                _current = null;
```

```
                return null;
            }
            _needEntry = false;
            _current = _contents.next();
            return JsonToken.FIELD_NAME;
        }
        _needEntry = true;
        return _current.getValue().asToken();
    }

    @Override
    public JsonToken nextValue()
    {
        JsonToken t = nextToken();
        if (t == JsonToken.FIELD_NAME) {
            t = nextToken();
        }
        return t;
    }

    @Override
    public JsonToken endToken() { return JsonToken.END_OBJECT; }

    @Override
    public JsonNode currentNode() {
        return (_current == null) ? null : _current.getValue();
    }
    @Override
    public boolean currentHasChildren() {
        // note: ONLY to be called for container nodes
        return ((ContainerNode) currentNode()).size() > 0;
    }
  }
}
package org.codehaus.jackson.node;

import java.io.IOException;
import java.util.Arrays;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Value node that contains Base64 encoded binary value, which will be
* output and stored as Json String value.
*/
public final class BinaryNode
  extends ValueNode
```

```
{
  final static BinaryNode EMPTY_BINARY_NODE = new BinaryNode(new byte[0]);

  final byte[] _data;

  public BinaryNode(byte[] data)
  {
     _data = data;
  }

  public BinaryNode(byte[] data, int offset, int length)
  {
     if (offset == 0 && length == data.length) {
        _data = data;
     } else {
        _data = new byte[length];
        System.arraycopy(data, offset, _data, 0, length);
     }
  }

  public static BinaryNode valueOf(byte[] data)
  {
     if (data == null) {
        return null;
     }
     if (data.length == 0) {
        return EMPTY_BINARY_NODE;
     }
     return new BinaryNode(data);
  }

  public static BinaryNode valueOf(byte[] data, int offset, int length)
  {
     if (data == null) {
        return null;
     }
     if (length == 0) {
        return EMPTY_BINARY_NODE;
     }
     return new BinaryNode(data, offset, length);
  }

  @Override
  public JsonToken asToken() {
     /* No distinct type; could use one for textual values,
      * but given that it's not in text form at this point,
      * embedded-object is closest
      */
```

```
    return JsonToken.VALUE_EMBEDDED_OBJECT;
}

@Override
public boolean isBinary() { return true; }

/**
 *<p>
 * Note: caller is not to modify returned array in any way, since
 * it is not a copy but reference to the underlying byte array.
 */
@Override
public byte[] getBinaryValue() { return _data; }

/**
 * Hmmh. This is not quite as efficient as using {@link #serialize},
 * but will work correctly.
 */
@Override
public String getValueAsText() {
    return Base64Variants.getDefaultVariant().encode(_data, false);
}

@Override
public final void serialize(JsonGenerator jg, SerializerProvider provider)
    throws IOException, JsonProcessingException
{
    jg.writeBinary(_data);
}

@Override
public boolean equals(Object o)
{
    if (o == this) return true;
    if (o == null) return false;
    if (o.getClass() != getClass()) { // final class, can do this
        return false;
    }
    return Arrays.equals(((BinaryNode) o)._data, _data);
}

@Override
public int hashCode() {
    return (_data == null) ? -1 : _data.length;
}

/**
 * Different from other values, since contents need to be surrounded
```

```
     * by (double) quotes.
     */
    @Override
    public String toString()
    {
        return Base64Variants.getDefaultVariant().encode(_data, true);
    }
}
package org.codehaus.jackson.node;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Numeric node that contains simple 64-bit integer values.
*/
public final class BigIntegerNode
    extends NumericNode
{
    final protected BigInteger _value;

    /*
    /**********************************************************
    /* Construction
    /**********************************************************
     */

    public BigIntegerNode(BigInteger v) { _value = v; }

    public static BigIntegerNode valueOf(BigInteger v) { return new BigIntegerNode(v); }

    /*
    /**********************************************************
    /* Overrridden JsonNode methods
    /**********************************************************
     */

    @Override
    public JsonToken asToken() { return JsonToken.VALUE_NUMBER_INT; }

    @Override
    public JsonParser.NumberType getNumberType() { return JsonParser.NumberType.BIG_INTEGER; }

    @Override
```

```
public boolean isIntegralNumber() { return true; }

@Override
public boolean isBigInteger() { return true; }

@Override
public Number getNumberValue() {
   return _value;
}

@Override
public int getIntValue() { return _value.intValue(); }

@Override
public long getLongValue() { return _value.longValue(); }

@Override
public BigInteger getBigIntegerValue() { return _value; }

@Override
public double getDoubleValue() { return _value.doubleValue(); }

@Override
public BigDecimal getDecimalValue() { return new BigDecimal(_value); }

/*
/**********************************************************
/* General type coercions
/**********************************************************
 */

@Override
public String getValueAsText() {
   return _value.toString();
}

@Override
public boolean getValueAsBoolean(boolean defaultValue) {
   return !BigInteger.ZERO.equals(_value);
}

@Override
public final void serialize(JsonGenerator jg, SerializerProvider provider)
   throws IOException, JsonProcessingException
{
   jg.writeNumber(_value);
}
```

```
  @Override
  public boolean equals(Object o)
  {
     if (o == this) return true;
     if (o == null) return false;
     if (o.getClass() != getClass()) { // final class, can do this
        return false;
     }
     return ((BigIntegerNode) o)._value == _value;
  }

  @Override
  public int hashCode() {
     return _value.hashCode();
  }
}
package org.codehaus.jackson.node;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.NumberOutput;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Numeric node that contains simple 64-bit integer values.
*/
public final class LongNode
  extends NumericNode
{
  final long _value;

  /*
  ************************************************
  * Construction
  ************************************************
  */

  public LongNode(long v) { _value = v; }

  public static LongNode valueOf(long l) { return new LongNode(l); }

  /*
  ************************************************
  * Overrridden JsonNode methods
  ************************************************
```

```
*/

@Override public JsonToken asToken() { return JsonToken.VALUE_NUMBER_INT; }

@Override
public JsonParser.NumberType getNumberType() { return JsonParser.NumberType.LONG; }


@Override
public boolean isIntegralNumber() { return true; }

@Override
public boolean isLong() { return true; }

@Override
public Number getNumberValue() {
   return Long.valueOf(_value);
}

@Override
public int getIntValue() { return (int) _value; }

@Override
public long getLongValue() { return _value; }

@Override
public double getDoubleValue() { return (double) _value; }

@Override
public BigDecimal getDecimalValue() { return BigDecimal.valueOf(_value); }

@Override
public BigInteger getBigIntegerValue() { return BigInteger.valueOf(_value); }

@Override
public String getValueAsText() {
   return NumberOutput.toString(_value);
}

@Override
public boolean getValueAsBoolean(boolean defaultValue) {
   return _value != 0;
}

@Override
public final void serialize(JsonGenerator jg, SerializerProvider provider)
   throws IOException, JsonProcessingException
{
```

```
        jg.writeNumber(_value);
    }

    @Override
    public boolean equals(Object o)
    {
        if (o == this) return true;
        if (o == null) return false;
        if (o.getClass() != getClass()) { // final class, can do this
            return false;
        }
        return ((LongNode) o)._value == _value;
    }

    @Override
    public int hashCode() {
        return ((int) _value) ^ (int) (_value >> 32);
    }
}
package org.codehaus.jackson.node;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.*;
import org.codehaus.jackson.impl.JsonParserMinimalBase;

/**
* Facade over {@link JsonNode} that implements {@link JsonParser} to allow
* accessing contents of JSON tree in alternate form (stream of tokens).
* Useful when a streaming source is expected by code, such as data binding
* functionality.
*
* @author tatu
*
* @since 1.3
*/
public class TreeTraversingParser extends JsonParserMinimalBase
{
    /*
    /**********************************************************
    /* Configuration
    /**********************************************************
     */

    protected ObjectCodec _objectCodec;
```

```
/**
 * Traversal context within tree
 */
protected NodeCursor _nodeCursor;

/*
/**********************************************************
/* State
/**********************************************************
 */

/**
 * Sometimes parser needs to buffer a single look-ahead token; if so,
 * it'll be stored here. This is currently used for handling
 */
protected JsonToken _nextToken;

/**
 * Flag needed to handle recursion into contents of child
 * Array/Object nodes.
 */
protected boolean _startContainer;

/**
 * Flag that indicates whether parser is closed or not. Gets
 * set when parser is either closed by explicit call
 * ({@link #close}) or when end-of-input is reached.
 */
protected boolean _closed;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public TreeTraversingParser(JsonNode n) { this(n, null); }

public TreeTraversingParser(JsonNode n, ObjectCodec codec)
{
   super(0);
   _objectCodec = codec;
   if (n.isArray()) {
      _nextToken = JsonToken.START_ARRAY;
      _nodeCursor = new NodeCursor.Array(n, null);
   } else if (n.isObject()) {
      _nextToken = JsonToken.START_OBJECT;
      _nodeCursor = new NodeCursor.Object(n, null);
```

```
    } else { // value node
        _nodeCursor = new NodeCursor.RootValue(n, null);
    }
}

@Override
public void setCodec(ObjectCodec c) {
    _objectCodec = c;
}

@Override
public ObjectCodec getCodec() {
    return _objectCodec;
}

/*
/**********************************************************
/* Closeable implementation
/**********************************************************
 */

@Override
public void close() throws IOException
{
    if (!_closed) {
        _closed = true;
        _nodeCursor = null;
        _currToken = null;
    }
}

/*
/**********************************************************
/* Public API, traversal
/**********************************************************
 */

@Override
public JsonToken nextToken() throws IOException, JsonParseException
{
    if (_nextToken != null) {
        _currToken = _nextToken;
        _nextToken = null;
        return _currToken;
    }
    // are we to descend to a container child?
    if (_startContainer) {
        _startContainer = false;
```

```
            // minor optimization: empty containers can be skipped
            if (!_nodeCursor.currentHasChildren()) {
                _currToken = (_currToken == JsonToken.START_OBJECT) ?
                    JsonToken.END_OBJECT : JsonToken.END_ARRAY;
                return _currToken;
            }
            _nodeCursor = _nodeCursor.iterateChildren();
            _currToken = _nodeCursor.nextToken();
            if (_currToken == JsonToken.START_OBJECT || _currToken == JsonToken.START_ARRAY) {
                _startContainer = true;
            }
            return _currToken;
        }
        // No more content?
        if (_nodeCursor == null) {
            _closed = true; // if not already set
            return null;
        }
        // Otherwise, next entry from current cursor
        _currToken = _nodeCursor.nextToken();
        if (_currToken != null) {
            if (_currToken == JsonToken.START_OBJECT || _currToken == JsonToken.START_ARRAY) {
                _startContainer = true;
            }
            return _currToken;
        }
        // null means no more children; need to return end marker
        _currToken = _nodeCursor.endToken();
        _nodeCursor = _nodeCursor.getParent();
        return _currToken;
    }

    // default works well here:
    //public JsonToken nextValue() throws IOException, JsonParseException

    @Override
    public JsonParser skipChildren() throws IOException, JsonParseException
    {
        if (_currToken == JsonToken.START_OBJECT) {
            _startContainer = false;
            _currToken = JsonToken.END_OBJECT;
        } else if (_currToken == JsonToken.START_ARRAY) {
            _startContainer = false;
            _currToken = JsonToken.END_ARRAY;
        }
        return this;
    }
```

```
@Override
public boolean isClosed() {
   return _closed;
}

/*
/**********************************************************
/* Public API, token accessors
/**********************************************************
 */

@Override
public String getCurrentName() {
   return (_nodeCursor == null) ? null : _nodeCursor.getCurrentName();
}

@Override
public JsonStreamContext getParsingContext() {
   return _nodeCursor;
}

@Override
public JsonLocation getTokenLocation() {
   return JsonLocation.NA;
}

@Override
public JsonLocation getCurrentLocation() {
   return JsonLocation.NA;
}

/*
/**********************************************************
/* Public API, access to textual content
/**********************************************************
 */

@Override
public String getText()
{
   if (_closed) {
      return null;
   }
   // need to separate handling a bit...
   switch (_currToken) {
   case FIELD_NAME:
      return _nodeCursor.getCurrentName();
   case VALUE_STRING:
```

```
            return currentNode().getTextValue();
        case VALUE_NUMBER_INT:
        case VALUE_NUMBER_FLOAT:
            return String.valueOf(currentNode().getNumberValue());
        case VALUE_EMBEDDED_OBJECT:
            JsonNode n = currentNode();
            if (n != null && n.isBinary()) {
                // this will convert it to base64
                return n.getValueAsText();
            }
        }

        return (_currToken == null) ? null : _currToken.asString();
    }

    @Override
    public char[] getTextCharacters() throws IOException, JsonParseException {
        return getText().toCharArray();
    }

    @Override
    public int getTextLength() throws IOException, JsonParseException {
        return getText().length();
    }

    @Override
    public int getTextOffset() throws IOException, JsonParseException {
        return 0;
    }

    @Override
    public boolean hasTextCharacters() {
        // generally we do not have efficient access as char[], hence:
        return false;
    }

    /*
    /**********************************************************
    /* Public API, typed non-text access
    /**********************************************************
     */

    //public byte getByteValue() throws IOException, JsonParseException

    @Override
    public NumberType getNumberType() throws IOException, JsonParseException {
        JsonNode n = currentNumericNode();
        return (n == null) ? null : n.getNumberType();
```

```
}

@Override
public BigInteger getBigIntegerValue() throws IOException, JsonParseException
{
   return currentNumericNode().getBigIntegerValue();
}

@Override
public BigDecimal getDecimalValue() throws IOException, JsonParseException {
   return currentNumericNode().getDecimalValue();
}

@Override
public double getDoubleValue() throws IOException, JsonParseException {
   return currentNumericNode().getDoubleValue();
}

@Override
public float getFloatValue() throws IOException, JsonParseException {
   return (float) currentNumericNode().getDoubleValue();
}

@Override
public long getLongValue() throws IOException, JsonParseException {
   return currentNumericNode().getLongValue();
}

@Override
public int getIntValue() throws IOException, JsonParseException {
   return currentNumericNode().getIntValue();
}

@Override
public Number getNumberValue() throws IOException, JsonParseException {
   return currentNumericNode().getNumberValue();
}

@Override
public Object getEmbeddedObject() {
   if (!_closed) {
      JsonNode n = currentNode();
      if (n != null && n.isPojo()) {
         return ((POJONode) n).getPojo();
      }
   }
   return null;
}
```

```
/*
/**********************************************************
/* Public API, typed binary (base64) access
/**********************************************************
 */

@Override
public byte[] getBinaryValue(Base64Variant b64variant)
   throws IOException, JsonParseException
{
   // Multiple possibilities...
   JsonNode n = currentNode();
   if (n != null) { // binary node?
      byte[] data = n.getBinaryValue();
      // (or TextNode, which can also convert automatically!)
      if (data != null) {
         return data;
      }
      // Or maybe byte[] as POJO?
      if (n.isPojo()) {
         Object ob = ((POJONode) n).getPojo();
         if (ob instanceof byte[]) {
            return (byte[]) ob;
         }
      }
   }
   // otherwise return null to mark we have no binary content
   return null;
}

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

protected JsonNode currentNode() {
   if (_closed || _nodeCursor == null) {
      return null;
   }
   return _nodeCursor.currentNode();
}

protected JsonNode currentNumericNode()
   throws JsonParseException
{
   JsonNode n = currentNode();
```

```
        if (n == null || !n.isNumber()) {
            JsonToken t = (n == null) ? null : n.asToken();
            throw _constructError("Current token ("+t+") not numeric, can not use numeric value accessors");
        }
        return n;
    }

    @Override
    protected void _handleEOF() throws JsonParseException {
        _throwInternal(); // should never get called
    }
}
package org.codehaus.jackson.node;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.SerializerProvider;

/**
* This singleton node class is generated to denote "missing nodes"
* along paths that do not exist. For example, if a path via
* element of an array is requested for an element outside range
* of elements in the array; or for a non-array value, result
* will be reference to this node.
*<p>
* In most respects this placeholder node will act as {@link NullNode};
* for example, for purposes of value conversions, value is considered
* to be null and represented as value zero when used for numeric
* conversions.
*/
public final class MissingNode
    extends BaseJsonNode
{
    private final static MissingNode instance = new MissingNode();

    private MissingNode() { }

    public static MissingNode getInstance() { return instance; }

    @Override public JsonToken asToken() { return JsonToken.NOT_AVAILABLE; }

    @Override
    public boolean isMissingNode() { return true; }

    @Override
    public String getValueAsText() { return null; }
```

```
@Override
public int getValueAsInt(int defaultValue) {
   return 0;
}
@Override
public long getValueAsLong(long defaultValue) {
   return 0L;
}
@Override
public double getValueAsDouble(double defaultValue) {
   return 0.0;
}

@Override
public JsonNode path(String fieldName) { return this; }

@Override
public JsonNode path(int index) { return this; }

@Override
public final void serialize(JsonGenerator jg, SerializerProvider provider)
   throws IOException, JsonProcessingException
{
   /* Nothing to output... should we signal an error tho?
    * Chances are, this is an erroneous call. For now, let's
    * not do that.
    */
}

@Override
public boolean equals(Object o)
{
   /* Hmmh. Since there's just a singleton instance, this
    * fails in all cases but with identity comparison.
    * However: if this placeholder value was to be considered
    * similar to Sql NULL, it shouldn't even equal itself?
    * That might cause problems when dealing with collections
    * like Sets... so for now, let's let identity comparison
    * return true.
    */
   return (o == this);
}

@Override
public String toString()
{
   // toString() should never return null
   return "";
```

```
  }
}
package org.codehaus.jackson.node;

import java.math.BigDecimal;
import java.math.BigInteger;

import org.codehaus.jackson.JsonParser;

/**
* Intermediate value node used for numeric nodes.
*/
public abstract class NumericNode
  extends ValueNode
{
  protected NumericNode() { }

  @Override
  public final boolean isNumber() { return true; }

  // // // Let's re-abstract so sub-classes handle them

  @Override
  public abstract JsonParser.NumberType getNumberType();

  @Override
  public abstract Number getNumberValue();
  @Override
  public abstract int getIntValue();
  @Override
  public abstract long getLongValue();
  @Override
  public abstract double getDoubleValue();
  @Override
  public abstract BigDecimal getDecimalValue();
  @Override
  public abstract BigInteger getBigIntegerValue();

  /*
  /**********************************************************
  /* General type coercions
  /**********************************************************
   */

  @Override
  public abstract String getValueAsText();

  @Override
```

```
  public int getValueAsInt() {
    return getIntValue();
  }
  @Override
  public int getValueAsInt(int defaultValue) {
    return getIntValue();
  }

  @Override
  public long getValueAsLong() {
    return getLongValue();
  }
  @Override
  public long getValueAsLong(long defaultValue) {
    return getLongValue();
  }

  @Override
  public double getValueAsDouble() {
    return getDoubleValue();
  }
  @Override
  public double getValueAsDouble(double defaultValue) {
    return getDoubleValue();
  }
}
package org.codehaus.jackson.node;

import java.io.IOException;
import java.math.BigDecimal;
import java.util.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Node that maps to JSON Object structures in JSON content.
*/
public class ObjectNode
  extends ContainerNode
{
  protected LinkedHashMap<String, JsonNode> _children = null;

  public ObjectNode(JsonNodeFactory nc) { super(nc); }

  /*
  /**********************************************************
  /* Implementation of core JsonNode API
```

```
/**********************************************************
 */

@Override public JsonToken asToken() { return JsonToken.START_OBJECT; }

@Override
public boolean isObject() { return true; }

@Override
public int size() {
   return (_children == null) ? 0 : _children.size();
}

@Override
public Iterator<JsonNode> getElements()
{
   return (_children == null) ? NoNodesIterator.instance() : _children.values().iterator();
}

@Override
public JsonNode get(int index) { return null; }

@Override
public JsonNode get(String fieldName)
{
   if (_children != null) {
      return _children.get(fieldName);
   }
   return null;
}

@Override
public Iterator<String> getFieldNames()
{
   return (_children == null) ? NoStringsIterator.instance() : _children.keySet().iterator();
}

@Override
public JsonNode path(int index)
{
   return MissingNode.getInstance();
}

@Override
public JsonNode path(String fieldName)
{
   if (_children != null) {
      JsonNode n = _children.get(fieldName);
```

```
        if (n != null) {
            return n;
        }
    }
    return MissingNode.getInstance();
}

/*
/**********************************************************
/* Public API, finding value nodes
/**********************************************************
 */

@Override
public JsonNode findValue(String fieldName)
{
    if (_children != null) {
        for (Map.Entry<String, JsonNode> entry : _children.entrySet()) {
            if (fieldName.equals(entry.getKey())) {
                return entry.getValue();
            }
            JsonNode value = entry.getValue().findValue(fieldName);
            if (value != null) {
                return value;
            }
        }
    }
    return null;
}

@Override
public List<JsonNode> findValues(String fieldName, List<JsonNode> foundSoFar)
{
    if (_children != null) {
        for (Map.Entry<String, JsonNode> entry : _children.entrySet()) {
            if (fieldName.equals(entry.getKey())) {
                if (foundSoFar == null) {
                    foundSoFar = new ArrayList<JsonNode>();
                }
                foundSoFar.add(entry.getValue());
            } else { // only add children if parent not added
                foundSoFar = entry.getValue().findValues(fieldName, foundSoFar);
            }
        }
    }
    return foundSoFar;
}
```

```
@Override
public List<String> findValuesAsText(String fieldName, List<String> foundSoFar)
{
   if (_children != null) {
      for (Map.Entry<String, JsonNode> entry : _children.entrySet()) {
         if (fieldName.equals(entry.getKey())) {
            if (foundSoFar == null) {
               foundSoFar = new ArrayList<String>();
            }
            foundSoFar.add(entry.getValue().getValueAsText());
         } else { // only add children if parent not added
            foundSoFar = entry.getValue().findValuesAsText(fieldName, foundSoFar);
         }
      }
   }
   return foundSoFar;
}

@Override
public ObjectNode findParent(String fieldName)
{
   if (_children != null) {
      for (Map.Entry<String, JsonNode> entry : _children.entrySet()) {
         if (fieldName.equals(entry.getKey())) {
            return this;
         }
         JsonNode value = entry.getValue().findParent(fieldName);
         if (value != null) {
            return (ObjectNode) value;
         }
      }
   }
   return null;
}

@Override
public List<JsonNode> findParents(String fieldName, List<JsonNode> foundSoFar)
{
   if (_children != null) {
      for (Map.Entry<String, JsonNode> entry : _children.entrySet()) {
         if (fieldName.equals(entry.getKey())) {
            if (foundSoFar == null) {
               foundSoFar = new ArrayList<JsonNode>();
            }
            foundSoFar.add(this);
         } else { // only add children if parent not added
            foundSoFar = entry.getValue().findParents(fieldName, foundSoFar);
         }
```

```
        }
    }
    return foundSoFar;
}

/*
/**********************************************************
/* Public API, serialization
/**********************************************************
 */

/**
 * Method that can be called to serialize this node and
 * all of its descendants using specified JSON generator.
 */
@Override
public final void serialize(JsonGenerator jg, SerializerProvider provider)
    throws IOException, JsonProcessingException
{
    jg.writeStartObject();
    if (_children != null) {
        for (Map.Entry<String, JsonNode> en : _children.entrySet()) {
            jg.writeFieldName(en.getKey());
            /* 17-Feb-2009, tatu: Can we trust that all nodes will always
             *   extend BaseJsonNode? Or if not, at least implement
             *   JsonSerializable? Let's start with former, change if
             *   we must.
             */
            ((BaseJsonNode) en.getValue()).serialize(jg, provider);
        }
    }
    jg.writeEndObject();
}

/*
/**********************************************************
/* Extended ObjectNode API, accessors
/**********************************************************
 */

/**
 * Method to use for accessing all fields (with both names
 * and values) of this Json Object.
 */
public Iterator<Map.Entry<String, JsonNode>> getFields()
{
    if (_children == null) {
        return NoFieldsIterator.instance;
```

```
    }
    return _children.entrySet().iterator();
}

/*
/**********************************************************
/* Extended ObjectNode API, mutators, generic
/**********************************************************
 */

/**
 * Method that will set specified field, replacing old value,
 * if any.
 *
 * @param value to set field to; if null, will be converted
 *   to a {@link NullNode} first  (to remove field entry, call
 *   {@link #remove} instead)
 *
 * @return Old value of the field, if any; null if there was no
 *   old value.
 */
public JsonNode put(String fieldName, JsonNode value)
{
    if (value == null) { // let's not store 'raw' nulls but nodes
        value = nullNode();
    }
    return _put(fieldName, value);
}

/**
 * Method for removing field entry from this ObjectNode.
 * Will return value of the field, if such field existed;
 * null if not.
 */
public JsonNode remove(String fieldName)
{
    if (_children != null) {
        return _children.remove(fieldName);
    }
    return null;
}

/**
 * Method for removing specified field properties out of
 * this ObjectNode.
 *
 * @param fieldNames Names of fields to remove
 *
```

```
     * @return This ObjectNode after removing entries
     *
     * @since 1.6
     */
    public ObjectNode remove(Collection<String> fieldNames)
    {
        if (_children != null) {
            for (String fieldName : fieldNames) {
                _children.remove(fieldName);
            }
        }
        return this;
    }

    /**
     * Method for removing all field properties, such that this
     * ObjectNode will contain no properties after call.
     */
    @Override
    public ObjectNode removeAll()
    {
        _children = null;
        return this;
    }

    /**
     * Method for adding given properties to this object node, overriding
     * any existing values for those properties.
     *
     * @param properties Properties to add
     *
     * @return This node (to allow chaining)
     *
     * @since 1.3
     */
    public JsonNode putAll(Map<String,JsonNode> properties)
    {
        if (_children == null) {
            _children = new LinkedHashMap<String, JsonNode>(properties);
        } else {
            for (Map.Entry<String, JsonNode> en : properties.entrySet()) {
                JsonNode n = en.getValue();
                if (n == null) {
                    n = nullNode();
                }
                _children.put(en.getKey(), n);
            }
        }
```

```
    return this;
}

/**
 * Method for adding all properties of the given Object, overriding
 * any existing values for those properties.
 *
 * @param other Object of which properties to add to this object
 *
 * @return This node (to allow chaining)
 *
 * @since 1.3
 */
public JsonNode putAll(ObjectNode other)
{
    int len = other.size();
    if (len > 0) {
        if (_children == null) {
            _children = new LinkedHashMap<String, JsonNode>(len);
        }
        other.putContentsTo(_children);
    }
    return this;
}

/**
 * Method for removing all field properties out of this ObjectNode
 * <b>except</b> for ones specified in argument.
 *
 * @param fieldNames Fields to <b>retain</b> in this ObjectNode
 *
 * @return This ObjectNode (to allow call chaining)
 *
 * @since 1.6
 */
public ObjectNode retain(Collection<String> fieldNames)
{
    if (_children != null) {
        Iterator<Map.Entry<String,JsonNode>> entries = _children.entrySet().iterator();
        while (entries.hasNext()) {
            Map.Entry<String, JsonNode> entry = entries.next();
            if (!fieldNames.contains(entry.getKey())) {
                entries.remove();
            }
        }
    }
    return this;
}
```

```
/**
 * Method for removing all field properties out of this ObjectNode
 * <b>except</b> for ones specified in argument.
 *
 * @param fieldNames Fields to <b>retain</b> in this ObjectNode
 *
 * @return This ObjectNode (to allow call chaining)
 *
 * @since 1.6
 */
public ObjectNode retain(String... fieldNames) {
   return retain(Arrays.asList(fieldNames));
}

/*
/**********************************************************
/* Extended ObjectNode API, mutators, typed
/**********************************************************
 */

/**
 * Method that will construct an ArrayNode and add it as a
 * field of this ObjectNode, replacing old value, if any.
 *
 * @return Newly constructed ArrayNode (NOT the old value,
 *   which could be of any type)
 */
public ArrayNode putArray(String fieldName)
{
   ArrayNode n  = arrayNode();
   _put(fieldName, n);
   return n;
}

/**
 * Method that will construct an ObjectNode and add it as a
 * field of this ObjectNode, replacing old value, if any.
 *
 * @return Newly constructed ObjectNode (NOT the old value,
 *   which could be of any type)
 */
public ObjectNode putObject(String fieldName)
{
   ObjectNode n  = objectNode();
   _put(fieldName, n);
   return n;
}
```

```
public void putPOJO(String fieldName, Object pojo)
{
    _put(fieldName, POJONode(pojo));
}

public void putNull(String fieldName)
{
    _put(fieldName, nullNode());
}

/**
 * Method for setting value of a field to specified numeric value.
 */
public void put(String fieldName, int v) { _put(fieldName, numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void put(String fieldName, long v) { _put(fieldName, numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void put(String fieldName, float v) { _put(fieldName, numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void put(String fieldName, double v) { _put(fieldName, numberNode(v)); }

/**
 * Method for setting value of a field to specified numeric value.
 */
public void put(String fieldName, BigDecimal v) {
    if (v == null) {
        putNull(fieldName);
    } else {
        _put(fieldName, numberNode(v));
    }
}

/**
 * Method for setting value of a field to specified String value.
 */
public void put(String fieldName, String v) {
    if (v == null) {
        putNull(fieldName);
```

```
    } else {
        _put(fieldName, textNode(v));
    }
}

/**
 * Method for setting value of a field to specified String value.
 */
public void put(String fieldName, boolean v) { _put(fieldName, booleanNode(v)); }

/**
 * Method for setting value of a field to specified binary value
 */
public void put(String fieldName, byte[] v) {
    if (v == null) {
        putNull(fieldName);
    } else {
        _put(fieldName, binaryNode(v));
    }
}

/*
/**********************************************************
/* Package methods (for other node classes to use)
/**********************************************************
 */

/**
 * @since 1.6
 */
protected void putContentsTo(Map<String,JsonNode> dst)
{
    if (_children != null) {
        for (Map.Entry<String,JsonNode> en : _children.entrySet()) {
            dst.put(en.getKey(), en.getValue());
        }
    }
}

/*
/**********************************************************
/* Standard methods
/**********************************************************
 */

@Override
public boolean equals(Object o)
{
```

```
    if (o == this) return true;
    if (o == null) return false;
    if (o.getClass() != getClass()) {
        return false;
    }
    ObjectNode other = (ObjectNode) o;
    if (other.size() != size()) {
        return false;
    }
    if (_children != null) {
        for (Map.Entry<String, JsonNode> en : _children.entrySet()) {
            String key = en.getKey();
            JsonNode value = en.getValue();

            JsonNode otherValue = other.get(key);

            if (otherValue == null || !otherValue.equals(value)) {
                return false;
            }
        }
    }
    return true;
}

@Override
public int hashCode()
{
    return (_children == null) ? -1 : _children.hashCode();
}

@Override
public String toString()
{
    StringBuilder sb = new StringBuilder(32 + (size() << 4));
    sb.append("{");
    if (_children != null) {
        int count = 0;
        for (Map.Entry<String, JsonNode> en : _children.entrySet()) {
            if (count > 0) {
                sb.append(",");
            }
            ++count;
            TextNode.appendQuoted(sb, en.getKey());
            sb.append(':');
            sb.append(en.getValue().toString());
        }
    }
    sb.append("}");
```

```
        return sb.toString();
    }

    /*
    /**********************************************************
    /* Internal methods
    /**********************************************************
     */

    private final JsonNode _put(String fieldName, JsonNode value)
    {
        if (_children == null) {
            _children = new LinkedHashMap<String, JsonNode>();
        }
        return _children.put(fieldName, value);
    }

    /*
    /**********************************************************
    /* Helper classes
    /**********************************************************
     */

    /**
     * For efficiency, let's share the "no fields" iterator...
     */
    protected static class NoFieldsIterator
        implements Iterator<Map.Entry<String, JsonNode>>
    {
        final static NoFieldsIterator instance = new NoFieldsIterator();

        private NoFieldsIterator() { }

        public boolean hasNext() { return false; }
        public Map.Entry<String,JsonNode> next() { throw new NoSuchElementException(); }

        public void remove() { // or IllegalOperationException?
            throw new IllegalStateException();
        }
    }
}
/**
Contains basic mapper (conversion) functionality that
allows for converting between regular streaming json content and
Java objects (beans or Tree Model: support for both is via
{@link org.codehaus.jackson.map.ObjectMapper} class, as well
as convenience methods included in
{@link org.codehaus.jackson.JsonParser}
```

<p>
Object mapper will convert Json content to ant from
basic Java wrapper types (Integer, Boolean, Double),
Collection types (List, Map), Java Beans,
Strings and nulls.
<p>
Tree mapper builds dynamically typed tree of <code>JsonNode</code>s
from Json content (and writes such trees as Json),
similar to how DOM model works with xml.
Main benefits over Object mapping are:

No null checks are needed (dummy
nodes are created as necessary to represent "missing" Object fields
and Array elements)

No type casts are usually needed: all public access methods are defined
in basic JsonNode class, and when "incompatible" method (such as Array
element access on, say, Boolean node) is used, returned node is
virtual "missing" node.

Because of its dynamic nature, Tree mapping is often convenient
for basic path access and tree navigation, where structure of
the resulting tree is known in advance.
*/

package org.codehaus.jackson.map;
package org.codehaus.jackson.map;

/**
* Add-on interface that {@link JsonSerializer}s can implement to get a callback
* that can be used to create contextual instances of serializer to use for
* handling properties of supported type. This can be useful
* for serializers that can be configured by annotations, or should otherwise
* have differing behavior depending on what kind of property is being serialized.
*
* @param <T> Type of serializer to contextualize
*
* @since 1.7
*/
public interface ContextualSerializer<T>
{
  /**
   * Method called to see if a different (or differently configured) serializer
   * is needed to serialize values of specified property.
   * Note that instance that this method is called on is typically shared one and
   * as a result method should NOT modify this instance but rather construct
   * and return a new instance. This instance should only be returned as-is, in case

```
     * it is already suitable for use.
     *
     * @param config Current serialization configuration
     * @param property Method or field that represents the property
     *   (and is used to access value to serialize).
     *   Should be available; but there may be cases where caller can not provide it and
     *   null is passed instead (in which case impls usually pass 'this' serializer as is)
     *
     * @return Serializer to use for serializing values of specified property;
     *   may be this instance or a new instance.
     *
     * @throws JsonMappingException
     */
    public JsonSerializer<T> createContextual(SerializationConfig config,
            BeanProperty property)
        throws JsonMappingException;
}
package org.codehaus.jackson.map;

import java.util.*;

import org.codehaus.jackson.map.introspect.AnnotatedMethod;
import org.codehaus.jackson.map.introspect.VisibilityChecker;
import org.codehaus.jackson.map.type.TypeBindings;
import org.codehaus.jackson.map.util.Annotations;
import org.codehaus.jackson.type.JavaType;

/**
* Basic container for information gathered by {@link ClassIntrospector} to
* help in constructing serializers and deserializers.
* Note that the main implementation type is
* {@link org.codehaus.jackson.map.introspect.BasicBeanDescription},
* meaning that it is safe to upcast to this type.
*
* @author tatu
*/
public abstract class BeanDescription
{
    /*
    /**********************************************************
    /* Configuration
    /**********************************************************
     */

    /**
     * Bean type information, including raw class and possible
     * * generics information
     */
```

```
protected final JavaType _type;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

protected BeanDescription(JavaType type)
{
 _type = type;
}

/*
/**********************************************************
/* Simple accesors
/**********************************************************
 */

/**
 * Method for accessing declared type of bean being introspected,
 * including full generic type information (from declaration)
 */
public JavaType getType() { return _type; }

public Class<?> getBeanClass() { return _type.getRawClass(); }

public abstract boolean hasKnownClassAnnotations();


/**
 * Accessor for type bindings that may be needed to fully resolve
 * types of member object, such as return and argument types of
 * methods and constructors, and types of fields.
 */
public abstract TypeBindings bindingsForBeanType();

/**
 * Method for accessing collection of annotations the bean
 * class has.
 *
 * @since 1.7
 */
public abstract Annotations getClassAnnotations();

/*
/**********************************************************
/* Basic API
```

```
  /**********************************************************
   */

  /**
   * @param visibilityChecker Object that determines whether
   *    methods have enough visibility to be auto-detectable as getters
   * @param ignoredProperties (optional, may be null) Names of properties
   *   to ignore; getters for these properties are not to be returned.
   *
   * @return Ordered Map with logical property name as key, and
   *    matching getter method as value.
   */
  public abstract LinkedHashMap<String,AnnotatedMethod> findGetters(VisibilityChecker<?> visibilityChecker,
        Collection<String> ignoredProperties);

  /**
   * @param vchecker (optional) Object that determines whether specific methods
   *   have enough visibility to be considered as auto-detectable setters.
   *   If null, auto-detection is disabled
   *
   * @return Ordered Map with logical property name as key, and
   *    matching setter method as value.
   */
  public abstract LinkedHashMap<String,AnnotatedMethod> findSetters(VisibilityChecker<?> vchecker);
}
package org.codehaus.jackson.map;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.type.*;
import org.codehaus.jackson.type.JavaType;

/**
* Interface that defines API for simple extensions that can provide additional deserializers
* for various types. Access is by a single callback method; instance is to either return
* a configured {@link JsonDeserializer} for specified type, or null to indicate that it
* does not support handling of the type. In latter case, further calls can be made
* for other providers; in former case returned deserializer is used for handling of
* instances of specified type.
*<p>
* Unlike with {@link Serializers}, multiple different methods are used since different
* kinds of types typically require different kinds of inputs.
*
* @since 1.7
*/
public interface Deserializers
{
  /**
   * Method called to locate constructor for specified array type.
```

```
*<p>
* Deserializer for element type may be passed, if configured explicitly at higher level (by
* annotations, typically), but usually are not.
* Type deserializer for element is passed if one is needed based on contextual information
* (annotations on declared element class; or on field or method type is associated with).
*
* @param type Type of array instances to deserialize
* @param config Configuration in effect
* @param provider Provider that can be used to locate deserializer for component type (if
*    one not provided, or needs to be overridden)
* @param property Property that contains array value  (null for root values)
* @param elementTypeDeserializer If element type needs polymorphic type handling, this is
*    the type information deserializer to use; should usually be used as is when constructing
*    array deserializer.
* @param elementDeserializer Deserializer to use for elements, if explicitly defined (by using
*    annotations, for exmple). May be null, in which case it should be resolved here (or using
*    {@link ResolvableDeserializer} callback)
*
* @return Deserializer to use for the type; or null if this provider does not know how to construct it
*/
public JsonDeserializer<?> findArrayDeserializer(ArrayType type, DeserializationConfig config,
       DeserializerProvider provider,
       BeanProperty property,
       TypeDeserializer elementTypeDeserializer, JsonDeserializer<?> elementDeserializer)
   throws JsonMappingException;

/**
* Method called to locate constructor for specified {@link java.util.Collection} (List, Set etc) type.
*<p>
* Deserializer for element type may be passed, if configured explicitly at higher level (by
* annotations, typically), but usually are not.
* Type deserializer for element is passed if one is needed based on contextual information
* (annotations on declared element class; or on field or method type is associated with).
*
* @param type Type of collection instances to deserialize
* @param config Configuration in effect
* @param provider Provider that can be used to locate dependant deserializers if and as necessary
*   (but note that in many cases resolution must be deferred by using {@link ResolvableDeserializer} callback)
* @param property Property that contains array value  (null for root values)
* @param beanDesc Definition of the enumeration type that contains class annotations and
*    other information typically needed for building deserializers (note: always instance
*    of {@link org.codehaus.jackson.map.introspect.BasicBeanDescription})
* @param elementTypeDeserializer If element type needs polymorphic type handling, this is
*    the type information deserializer to use; should usually be used as is when constructing
*    array deserializer.
* @param elementDeserializer Deserializer to use for elements, if explicitly defined (by using
*    annotations, for exmple). May be null, in which case it should be resolved here (or using
*    {@link ResolvableDeserializer} callback)
```

```
     *
     * @return Deserializer to use for the type; or null if this provider does not know how to construct it
     */
    public JsonDeserializer<?> findCollectionDeserializer(CollectionType type, DeserializationConfig config,
            DeserializerProvider provider,
            BeanDescription beanDesc,
            BeanProperty property,
            TypeDeserializer elementTypeDeserializer, JsonDeserializer<?> elementDeserializer)
        throws JsonMappingException;

    /**
     * Method called to locate deserializer for specified {@link java.lang.Enum} type.
     *
     * @param type Type of {@link java.lang.Enum} instances to deserialize
     * @param config Configuration in effect
     * @param beanDesc Definition of the enumeration type that contains class annotations and
     *    other information typically needed for building deserializers (note: always instance
     *    of {@link org.codehaus.jackson.map.introspect.BasicBeanDescription})
     *
     * @return Deserializer to use for the type; or null if this provider does not know how to construct it
     */
    public JsonDeserializer<?> findEnumDeserializer(Class<?> type, DeserializationConfig config,
            BeanDescription beanDesc,
            BeanProperty property)
        throws JsonMappingException;

    /**
     * Method called to locate deserializer for specified {@link java.util.Map} type.
     *<p>
     * Deserializer for element type may be passed, if configured explicitly at higher level (by
     * annotations, typically), but usually are not.
     * Type deserializer for element is passed if one is needed based on contextual information
     * (annotations on declared element class; or on field or method type is associated with).
     *<p>
     * Similarly, a {@link KeyDeserializer} may be passed, but this is only done if there is
     * a specific configuration override (annotations) to indicate instance to use. Otherwise
     * null is passed, and key deserializer needs to be obtained using {@link DeserializerProvider}
     *
     * @param type Type of {@link java.util.Map} instances to deserialize
     * @param config Configuration in effect
     * @param provider Provider that can be used to locate dependant deserializers if and as necessary
     *   (but note that in many cases resolution must be deferred by using {@link ResolvableDeserializer} callback)
     * @param beanDesc Definition of the enumeration type that contains class annotations and
     *    other information typically needed for building deserializers (note: always instance
     *    of {@link org.codehaus.jackson.map.introspect.BasicBeanDescription})
     * @param keyDeserializer Key deserializer use, if it is defined via annotations or other configuration;
     *    null if default key deserializer for key type can be used.
     * @param elementTypeDeserializer If element type needs polymorphic type handling, this is
```

```
     *    the type information deserializer to use; should usually be used as is when constructing
     *    array deserializer.
     * @param elementDeserializer Deserializer to use for elements, if explicitly defined (by using
     *    annotations, for exmple). May be null, in which case it should be resolved here (or using
     *    {@link ResolvableDeserializer} callback)
     *
     * @return Deserializer to use for the type; or null if this provider does not know how to construct it
     */
    public JsonDeserializer<?> findMapDeserializer(MapType type, DeserializationConfig config,
            DeserializerProvider provider,
            BeanDescription beanDesc,
            BeanProperty property,
            KeyDeserializer keyDeserializer,
            TypeDeserializer elementTypeDeserializer, JsonDeserializer<?> elementDeserializer)
        throws JsonMappingException;

    /**
     * Method called to locate deserializer for specified JSON tree node type.
     *
     * @param nodeType Specific type of JSON tree nodes to deserialize (subtype of {@link
org.codehaus.jackson.JsonNode})
     * @param config Configuration in effect
     *
     * @return Deserializer to use for the type; or null if this provider does not know how to construct it
     */
    public JsonDeserializer<?> findTreeNodeDeserializer(Class<? extends JsonNode> nodeType,
DeserializationConfig config,
            BeanProperty property)
        throws JsonMappingException;

    /**
     * Method called to locate deserializer for specified value type which does not belong to any other
     * category (not an Enum, Collection, Map, Array or tree node)
     *
     * @param type Bean type to deserialize
     * @param config Configuration in effect
     * @param provider Provider that can be used to locate dependant deserializers if and as necessary
     *   (but note that in many cases resolution must be deferred by using {@link ResolvableDeserializer} callback)
     * @param beanDesc Definition of the enumeration type that contains class annotations and
     *    other information typically needed for building deserializers (note: always instance
     *    of {@link org.codehaus.jackson.map.introspect.BasicBeanDescription})
     *
     * @return Deserializer to use for the type; or null if this provider does not know how to construct it
     */
    public JsonDeserializer<?> findBeanDeserializer(JavaType type, DeserializationConfig config,
            DeserializerProvider provider, BeanDescription beanDesc,
            BeanProperty property)
        throws JsonMappingException;
```

```
/*
/**********************************************************
/* Helper classes
/**********************************************************
 */

/**
 * Basic {@link Deserializers} implementation that implements all methods but provides
 * no deserializers. Its main purpose is to serve as a base class so that
 * sub-classes only need to override methods they need, as most of the time some
 * of methods are not needed (especially enumeration and array deserializers are
 * very rarely overridde).
 *
 * @since 1.7
 */
public static class None implements Deserializers
{
   @Override
   public JsonDeserializer<?> findArrayDeserializer(ArrayType type, DeserializationConfig config,
         DeserializerProvider provider,
         BeanProperty property,
         TypeDeserializer elementTypeDeserializer, JsonDeserializer<?> elementDeserializer)
      throws JsonMappingException
   {
      return null;
   }

   @Override
   public JsonDeserializer<?> findCollectionDeserializer(CollectionType type, DeserializationConfig config,
         DeserializerProvider provider,
         BeanDescription beanDesc,
         BeanProperty property,
         TypeDeserializer elementTypeDeserializer, JsonDeserializer<?> elementDeserializer)
      throws JsonMappingException
   {
      return null;
   }

   @Override
   public JsonDeserializer<?> findEnumDeserializer(Class<?> type, DeserializationConfig config,
         BeanDescription beanDesc,
         BeanProperty property)
      throws JsonMappingException
   {
      return null;
   }
```

```
    @Override
    public JsonDeserializer<?> findMapDeserializer(MapType type, DeserializationConfig config,
            DeserializerProvider provider,
            BeanDescription beanDesc,
            BeanProperty property,
            KeyDeserializer keyDeserializer,
            TypeDeserializer elementTypeDeserializer, JsonDeserializer<?> elementDeserializer)
    throws JsonMappingException
    {
       return null;
    }

    @Override
    public JsonDeserializer<?> findTreeNodeDeserializer(Class<? extends JsonNode> nodeType,
            DeserializationConfig config,
            BeanProperty property)
       throws JsonMappingException
    {
       return null;
    }

    @Override
    public JsonDeserializer<?> findBeanDeserializer(JavaType type, DeserializationConfig config,
            DeserializerProvider provider,
            BeanDescription beanDesc,
            BeanProperty property)
       throws JsonMappingException
    {
       return null;
    }

  }
}
package org.codehaus.jackson.map;

import java.io.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.SegmentedStringWriter;
import org.codehaus.jackson.map.introspect.VisibilityChecker;
import org.codehaus.jackson.map.jsontype.SubtypeResolver;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.map.ser.FilterProvider;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.type.TypeReference;
import org.codehaus.jackson.util.ByteArrayBuilder;
import org.codehaus.jackson.util.DefaultPrettyPrinter;
```

```
import org.codehaus.jackson.util.MinimalPrettyPrinter;
import org.codehaus.jackson.util.VersionUtil;

/**
* Builder object that can be used for per-serialization configuration of
* serialization parameters, such as JSON View and root type to use.
* Uses "fluent" (aka builder) pattern so that instances are immutable
* (and thus fully thread-safe with no external synchronization);
* new instances are constructed for different configurations.
* Instances are initially constructed by {@link ObjectMapper} and can be
* reused in completely thread-safe manner with no explicit synchronization
*
* @author tatu
* @since 1.5
*/
public class ObjectWriter
  implements Versioned // since 1.6
{
  /**
   * We need to keep track of explicit disabling of pretty printing;
   * easiest to do by a token value.
   */
  protected final static PrettyPrinter NULL_PRETTY_PRINTER = new MinimalPrettyPrinter();

  /*
  /**********************************************************
  /* Immutable configuration from ObjectMapper
  /**********************************************************
   */

  /**
   * General serialization configuration settings
   */
  protected final SerializationConfig _config;

  protected final SerializerProvider _provider;

  protected final SerializerFactory _serializerFactory;

  /**
   * Factory used for constructing {@link JsonGenerator}s
   */
  protected final JsonFactory _jsonFactory;

  // Support for polymorphic types:
  protected final TypeResolverBuilder<?> _defaultTyper;

  // Configurable visibility limits
```

```
protected final VisibilityChecker<?> _visibilityChecker;

/**
 * Registered concrete subtypes that can be used instead of (or
 * in addition to) ones declared using annotations.
 *
 * @since 1.6
 */
protected final SubtypeResolver _subtypeResolver;

/*
/**********************************************************
/* Configuration that can be changed during building
/**********************************************************
 */

/**
 * View to use for serialization
 */
protected final Class<?> _serializationView;

/**
 * Specified root serialization type to use; can be same
 * as runtime type, but usually one of its super types
 */
protected final JavaType _rootType;

/**
 * To allow for dynamic enabling/disabling of pretty printing,
 * pretty printer can be optionally configured for writer
 * as well
 */
protected final PrettyPrinter _prettyPrinter;

/*
/**********************************************************
/* Life-cycle, constructors
/**********************************************************
 */

/**
 * Constructor used by {@link ObjectMapper} for initial instantiation
 */
protected ObjectWriter(ObjectMapper mapper,
        Class<?> view, JavaType rootType, PrettyPrinter pp)
{
    _defaultTyper = mapper._defaultTyper;
    _visibilityChecker = mapper._visibilityChecker;
```

```
    _subtypeResolver = mapper._subtypeResolver;
    // must make a copy at this point, to prevent further changes from trickling down
    _config = mapper._serializationConfig.createUnshared(_defaultTyper, _visibilityChecker,
            _subtypeResolver, null);
    _config.setSerializationView(view);

    _provider = mapper._serializerProvider;
    _serializerFactory = mapper._serializerFactory;

    _jsonFactory = mapper._jsonFactory;

    _serializationView = view;
    _rootType = rootType;
    _prettyPrinter = pp;
}

/**
 * Alternative constructor for initial instantiation; used when a filter provider
 * is given.
 *
 * @since 1.7
 */
protected ObjectWriter(ObjectMapper mapper, FilterProvider filterProvider)
{
    _defaultTyper = mapper._defaultTyper;
    _visibilityChecker = mapper._visibilityChecker;
    _subtypeResolver = mapper._subtypeResolver;
    // must make a copy at this point, to prevent further changes from trickling down
    _config = mapper._serializationConfig.createUnshared(_defaultTyper, _visibilityChecker,
            _subtypeResolver, filterProvider);
    _provider = mapper._serializerProvider;
    _serializerFactory = mapper._serializerFactory;
    _jsonFactory = mapper._jsonFactory;
    _serializationView = null;
    _rootType = null;
    _prettyPrinter = null;
}

/**
 * Copy constructor used for building variations.
 */
protected ObjectWriter(ObjectWriter base, SerializationConfig config,
        Class<?> view, JavaType rootType, PrettyPrinter pp)
{
    _config = config;
    _provider = base._provider;
    _serializerFactory = base._serializerFactory;
```

```
    _jsonFactory = base._jsonFactory;
    _defaultTyper = base._defaultTyper;
    _visibilityChecker = base._visibilityChecker;
    _subtypeResolver = base._subtypeResolver;

    _serializationView = view;
    _rootType = rootType;
    _prettyPrinter = pp;
}

/**
 * Copy constructor used for building variations.
 *
 * @since 1.7
 */
protected ObjectWriter(ObjectWriter base, SerializationConfig config)
{
    _config = config;

    _provider = base._provider;
    _serializerFactory = base._serializerFactory;

    _jsonFactory = base._jsonFactory;
    _defaultTyper = base._defaultTyper;
    _visibilityChecker = base._visibilityChecker;
    _subtypeResolver = base._subtypeResolver;

    _serializationView = base._serializationView;
    _rootType = base._rootType;
    _prettyPrinter = base._prettyPrinter;

}

/**
 * Method that will return version information stored in and read from jar
 * that contains this class.
 *
 * @since 1.6
 */
public Version version() {
    return VersionUtil.versionFor(getClass());
}

/*
/**********************************************************
/* Life-cycle, fluent factories
/**********************************************************
 */
```

```
/**
 * Method that will construct a new instance that uses specified
 * serialization view for serialization (with null basically disables
 * view processing)
 */
public ObjectWriter withView(Class<?> view)
{
   if (view == _serializationView) return this;
   // View is included in config, must make immutable version
   SerializationConfig config = _config.createUnshared(_defaultTyper,
         _visibilityChecker, _subtypeResolver);
   config.setSerializationView(view);
   return new ObjectWriter(this, config);
}

/**
 * Method that will construct a new instance that uses specific type
 * as the root type for serialization, instead of runtime dynamic
 * type of the root object itself.
 */
public ObjectWriter withType(JavaType rootType)
{
   if (rootType == _rootType) return this;
   // type is stored here, no need to make a copy of config
   return new ObjectWriter(this, _config, _serializationView, rootType, _prettyPrinter);
}

/**
 * Method that will construct a new instance that uses specific type
 * as the root type for serialization, instead of runtime dynamic
 * type of the root object itself.
 */
public ObjectWriter withType(Class<?> rootType)
{
   return withType(TypeFactory.type(rootType));
}

/**
 * @since 1.7
 */
public ObjectWriter withType(TypeReference<?> rootType)
{
   return withType(TypeFactory.type(rootType));
}

/**
 * Method that will construct a new instance that will use specified pretty
```

```
     * printer (or, if null, will not do any pretty-printing)
     *
     * @since 1.6
     */
    public ObjectWriter withPrettyPrinter(PrettyPrinter pp)
    {
        // since null would mean "don't care", need to use placeholder to indicate "disable"
        if (pp == null) {
            pp = NULL_PRETTY_PRINTER;
        }
        return new ObjectWriter(this, _config, _serializationView, _rootType, pp);
    }

    /**
     * Method that will construct a new instance that will use the default
     * pretty printer for serialization.
     *
     * @since 1.6
     */
    public ObjectWriter withDefaultPrettyPrinter()
    {
        return withPrettyPrinter(new DefaultPrettyPrinter());
    }

    /**
     * Method that will construct a new instance that uses specified
     * provider for resolving filter instances by id.
     *
     * @since 1.7
     */
    public ObjectWriter withFilters(FilterProvider filterProvider) {
        if (filterProvider == _config.getFilterProvider()) { // no change?
            return this;
        }
        return new ObjectWriter(this, _config.withFilters(filterProvider));
    }

    /*
    /**********************************************************
    /* Serialization methods; ones from ObjectCodec first
    /**********************************************************
     */

    /**
     * Method that can be used to serialize any Java value as
     * JSON output, using provided {@link JsonGenerator}.
     */
    public void writeValue(JsonGenerator jgen, Object value)
```

```
    throws IOException, JsonGenerationException, JsonMappingException
{
    if (_config.isEnabled(SerializationConfig.Feature.CLOSE_CLOSEABLE) && (value instanceof Closeable)) {
        _writeCloseableValue(jgen, value, _config);
    } else {
        if (_rootType == null) {
            _provider.serializeValue(_config, jgen, value, _serializerFactory);
        } else {
            _provider.serializeValue(_config, jgen, value, _rootType, _serializerFactory);
        }
        if (_config.isEnabled(SerializationConfig.Feature.FLUSH_AFTER_WRITE_VALUE)) {
            jgen.flush();
        }
    }
}

/*
/**********************************************************
/* Serialization methods, others
/**********************************************************
 */

/**
 * Method that can be used to serialize any Java value as
 * JSON output, written to File provided.
 */
public void writeValue(File resultFile, Object value)
    throws IOException, JsonGenerationException, JsonMappingException
{
    _configAndWriteValue(_jsonFactory.createJsonGenerator(resultFile, JsonEncoding.UTF8), value);
}

/**
 * Method that can be used to serialize any Java value as
 * JSON output, using output stream provided (using encoding
 * {@link JsonEncoding#UTF8}).
 *<p>
 * Note: method does not close the underlying stream explicitly
 * here; however, {@link JsonFactory} this mapper uses may choose
 * to close the stream depending on its settings (by default,
 * it will try to close it when {@link JsonGenerator} we construct
 * is closed).
 */
public void writeValue(OutputStream out, Object value)
    throws IOException, JsonGenerationException, JsonMappingException
{
    _configAndWriteValue(_jsonFactory.createJsonGenerator(out, JsonEncoding.UTF8), value);
}
```

```
/**
 * Method that can be used to serialize any Java value as
 * JSON output, using Writer provided.
 *<p>
 * Note: method does not close the underlying stream explicitly
 * here; however, {@link JsonFactory} this mapper uses may choose
 * to close the stream depending on its settings (by default,
 * it will try to close it when {@link JsonGenerator} we construct
 * is closed).
 */
public void writeValue(Writer w, Object value)
   throws IOException, JsonGenerationException, JsonMappingException
{
   _configAndWriteValue(_jsonFactory.createJsonGenerator(w), value);
}

/**
 * Method that can be used to serialize any Java value as
 * a String. Functionally equivalent to calling
 * {@link #writeValue(Writer,Object)} with {@link java.io.StringWriter}
 * and constructing String, but more efficient.
 */
public String writeValueAsString(Object value)
   throws IOException, JsonGenerationException, JsonMappingException
{
   // alas, we have to pull the recycler directly here...
   SegmentedStringWriter sw = new SegmentedStringWriter(_jsonFactory._getBufferRecycler());
   _configAndWriteValue(_jsonFactory.createJsonGenerator(sw), value);
   return sw.getAndClear();
}

/**
 * Method that can be used to serialize any Java value as
 * a byte array. Functionally equivalent to calling
 * {@link #writeValue(Writer,Object)} with {@link java.io.ByteArrayOutputStream}
 * and getting bytes, but more efficient.
 * Encoding used will be UTF-8.
 */
public byte[] writeValueAsBytes(Object value)
   throws IOException, JsonGenerationException, JsonMappingException
{
   ByteArrayBuilder bb = new ByteArrayBuilder(_jsonFactory._getBufferRecycler());
   _configAndWriteValue(_jsonFactory.createJsonGenerator(bb, JsonEncoding.UTF8), value);
   byte[] result = bb.toByteArray();
   bb.release();
   return result;
}
```

```
/*
/**********************************************************
/* Other public methods
/**********************************************************
 */

public boolean canSerialize(Class<?> type)
{
   return _provider.hasSerializerFor(_config, type, _serializerFactory);
}

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

/**
 * Method called to configure the generator as necessary and then
 * call write functionality
 */
protected final void _configAndWriteValue(JsonGenerator jgen, Object value)
   throws IOException, JsonGenerationException, JsonMappingException
{
   if (_prettyPrinter != null) {
      PrettyPrinter pp = _prettyPrinter;
      jgen.setPrettyPrinter((pp == NULL_PRETTY_PRINTER) ? null : pp);
   } else if (_config.isEnabled(SerializationConfig.Feature.INDENT_OUTPUT)) {
      jgen.useDefaultPrettyPrinter();
   }
   // [JACKSON-282]: consider Closeable
   if (_config.isEnabled(SerializationConfig.Feature.CLOSE_CLOSEABLE) && (value instanceof Closeable)) {
      _configAndWriteCloseable(jgen, value, _config);
      return;
   }
   boolean closed = false;
   try {
      if (_rootType == null) {
         _provider.serializeValue(_config, jgen, value, _serializerFactory);
      } else {
         _provider.serializeValue(_config, jgen, value, _rootType, _serializerFactory);
      }
      closed = true;
      jgen.close();
   } finally {
      /* won't try to close twice; also, must catch exception (so it
       * will not mask exception that is pending)
```

```
         */
        if (!closed) {
            try {
                jgen.close();
            } catch (IOException ioe) { }
        }
    }
}

/**
 * Helper method used when value to serialize is {@link Closeable} and its <code>close()</code>
 * method is to be called right after serialization has been called
 */
private final void _configAndWriteCloseable(JsonGenerator jgen, Object value, SerializationConfig cfg)
    throws IOException, JsonGenerationException, JsonMappingException
{
    Closeable toClose = (Closeable) value;
    try {
        if (_rootType == null) {
            _provider.serializeValue(cfg, jgen, value, _serializerFactory);
        } else {
            _provider.serializeValue(cfg, jgen, value, _rootType, _serializerFactory);
        }
        JsonGenerator tmpJgen = jgen;
        jgen = null;
        tmpJgen.close();
        Closeable tmpToClose = toClose;
        toClose = null;
        tmpToClose.close();
    } finally {
        /* Need to close both generator and value, as long as they haven't yet
         * been closed
         */
        if (jgen != null) {
            try {
                jgen.close();
            } catch (IOException ioe) { }
        }
        if (toClose != null) {
            try {
                toClose.close();
            } catch (IOException ioe) { }
        }
    }
}

/**
 * Helper method used when value to serialize is {@link Closeable} and its <code>close()</code>
```

```
     * method is to be called right after serialization has been called
     */
    private final void _writeCloseableValue(JsonGenerator jgen, Object value, SerializationConfig cfg)
        throws IOException, JsonGenerationException, JsonMappingException
    {
        Closeable toClose = (Closeable) value;
        try {
            if (_rootType == null) {
                _provider.serializeValue(cfg, jgen, value, _serializerFactory);
            } else {
                _provider.serializeValue(cfg, jgen, value, _rootType, _serializerFactory);
            }
            if (_config.isEnabled(SerializationConfig.Feature.FLUSH_AFTER_WRITE_VALUE)) {
                jgen.flush();
            }
            Closeable tmpToClose = toClose;
            toClose = null;
            tmpToClose.close();
        } finally {
            if (toClose != null) {
                try {
                    toClose.close();
                } catch (IOException ioe) { }
            }
        }
    }
}
package org.codehaus.jackson.map;

import java.text.DateFormat;
import java.util.*;

import org.codehaus.jackson.annotate.*;
import org.codehaus.jackson.map.annotate.JsonSerialize;
import org.codehaus.jackson.map.annotate.JsonSerialize.Inclusion; // for javadocs
import org.codehaus.jackson.map.introspect.AnnotatedClass;
import org.codehaus.jackson.map.introspect.VisibilityChecker;
import org.codehaus.jackson.map.jsontype.SubtypeResolver;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.map.jsontype.impl.StdSubtypeResolver;
import org.codehaus.jackson.map.ser.FilterProvider;
import org.codehaus.jackson.map.type.ClassKey;
import org.codehaus.jackson.map.util.StdDateFormat;
import org.codehaus.jackson.type.JavaType;

/**
* Object that contains baseline configuration for serialization
* process. An instance is owned by {@link ObjectMapper}, which makes
```

```
* a copy that is passed during serialization process to
* {@link SerializerProvider} and {@link SerializerFactory}.
*<p>
* Note: although configuration settings can be changed at any time
* (for factories and instances), they are not guaranteed to have
* effect if called after constructing relevant mapper or serializer
* instance. This because some objects may be configured, constructed and
* cached first time they are needed.
*/
public class SerializationConfig
  implements MapperConfig<SerializationConfig>
{
  /**
   * Enumeration that defines togglable features that guide
   * the serialization feature.
   */
  public enum Feature {
    /*
    /*********************************************************
     *  Introspection features
    /*********************************************************
     */

    /**
     * Feature that determines whether annotation introspection
     * is used for configuration; if enabled, configured
     * {@link AnnotationIntrospector} will be used: if disabled,
     * no annotations are considered.
     *<P>
     * Feature is enabled by default.
     *
     * @since 1.2
     */
    USE_ANNOTATIONS(true),

    /**
     * Feature that determines whether regualr "getter" methods are
     * automatically detected based on standard Bean naming convention
     * or not. If yes, then all public zero-argument methods that
     * start with prefix "get"
     * are considered as getters.
     * If disabled, only methods explicitly  annotated are considered getters.
     *<p>
     * Note that since version 1.3, this does <b>NOT</b> include
     * "is getters" (see {@link #AUTO_DETECT_IS_GETTERS} for details)
     *<p>
     * Note that this feature has lower precedence than per-class
     * annotations, and is only used if there isn't more granular
```

```
 * configuration available.
 *<P>
 * Feature is enabled by default.
 */
AUTO_DETECT_GETTERS(true),

/**
 * Feature that determines whether "is getter" methods are
 * automatically detected based on standard Bean naming convention
 * or not. If yes, then all public zero-argument methods that
 * start with prefix "is", and whose return type is boolean
 * are considered as "is getters".
 * If disabled, only methods explicitly annotated are considered getters.
 *<p>
 * Note that this feature has lower precedence than per-class
 * annotations, and is only used if there isn't more granular
 * configuration available.
 *<P>
 * Feature is enabled by default.
 */
AUTO_DETECT_IS_GETTERS(true),

/**
 * Feature that determines whether non-static fields are recognized as
 * properties.
 * If yes, then all public member fields
 * are considered as properties. If disabled, only fields explicitly
 * annotated are considered property fields.
 *<p>
 * Note that this feature has lower precedence than per-class
 * annotations, and is only used if there isn't more granular
 * configuration available.
 *<P>
 * Feature is enabled by default.
 *
 * @since 1.1
 */
 AUTO_DETECT_FIELDS(true),

/**
 * Feature that determines whether method and field access
 * modifier settings can be overridden when accessing
 * properties. If enabled, method
 * {@link java.lang.reflect.AccessibleObject#setAccessible}
 * may be called to enable access to otherwise unaccessible
 * objects.
 */
CAN_OVERRIDE_ACCESS_MODIFIERS(true),
```

```
/*
/**********************************************************
 * Generic output features
/**********************************************************
 */

/**
 * Feature that determines the default settings of whether Bean
 * properties with null values are to be written out.
 *<p>
 * Feature is enabled by default (null properties written).
 *<p>
 * Note too that there is annotation
 * {@link org.codehaus.jackson.annotate.JsonWriteNullProperties}
 * that can be used for more granular control (annotates bean
 * classes or individual property access methods).
 *
 * @deprecated As of 1.1, use {@link SerializationConfig#setSerializationInclusion}
 *    instead
 */
@Deprecated
WRITE_NULL_PROPERTIES(true),

/**
 * Feature that determines whether the type detection for
 * serialization should be using actual dynamic runtime type,
 * or declared static type.
 * Default value is false, to use dynamic runtime type.
 *<p>
 * This global default value can be overridden at class, method
 * or field level by using {@link JsonSerialize#typing} annotation
 * property
 */
USE_STATIC_TYPING(false),

/**
 * Feature that determines whether properties that have no view
 * annotations are included in JSON serialization views (see
 * {@link org.codehaus.jackson.map.annotate.JsonView} for more
 * details on JSON Views).
 * If enabled, non-annotated properties will be included;
 * when disabled, they will be excluded. So this feature
 * changes between "opt-in" (feature disabled) and
 * "opt-out" (feature enabled) modes.
 *<p>
 * Default value is enabled, meaning that non-annotated
 * properties are included in all views if there is no
```

```
     * {@link org.codehaus.jackson.map.annotate.JsonView} annotation.
     *
     * @since 1.5
     */
    DEFAULT_VIEW_INCLUSION(true),

    /**
     * Feature that can be enabled to make root value (usually JSON
     * Object but can be any type) wrapped within a single property
     * JSON object, where key as the "root name", as determined by
     * annotation introspector (esp. for JAXB that uses
     * <code>@XmlRootElement.name</code>) or fallback (non-qualified
     * class name).
     * Feature is mostly intended for JAXB compatibility.
     *<p>
     * Default setting is false, meaning root value is not wrapped.
     *
     * @since 1.7
     */
    WRAP_ROOT_VALUE(false),

    /**
     * Feature that allows enabling (or disabling) indentation
     * for the underlying generator, using the default pretty
     * printer (see
     * {@link org.codehaus.jackson.JsonGenerator#useDefaultPrettyPrinter}
     * for details).
     *<p>
     * Note that this only affects cases where
     * {@link org.codehaus.jackson.JsonGenerator}
     * is constructed implicitly by ObjectMapper: if explicit
     * generator is passed, its configuration is not changed.
     *<p>
     * Also note that if you want to configure details of indentation,
     * you need to directly configure the generator: there is a
     * method to use any <code>PrettyPrinter</code> instance.
     * This feature will only allow using the default implementation.
     */
    INDENT_OUTPUT(false),

    /*
    /******************************************************
     *  Error handling features
    /******************************************************
     */

    /**
     * Feature that determines what happens when no accessors are
```

```
     * found for a type (and there are no annotations to indicate
     * it is meant to be serialized). If enabled (default), an
     * exception is thrown to indicate these as non-serializable
     * types; if disabled, they are serialized as empty Objects,
     * i.e. without any properties.
     *<p>
     * Note that empty types that this feature has only effect on
     * those "empty" beans that do not have any recognized annotations
     * (like <code>@JsonSerialize</code>): ones that do have annotations
     * do not result in an exception being thrown.
     *
     * @since 1.4
     */
    FAIL_ON_EMPTY_BEANS(true),

    /**
     * Feature that determines whether Jackson code should catch
     * and wrap {@link Exception}s (but never {@link Error}s!)
     * to add additional information about
     * location (within input) of problem or not. If enabled,
     * most exceptions will be caught and re-thrown (exception
     * specifically being that {@link java.io.IOException}s may be passed
     * as is, since they are declared as throwable); this can be
     * convenient both in that all exceptions will be checked and
     * declared, and so there is more contextual information.
     * However, sometimes calling application may just want "raw"
     * unchecked exceptions passed as is.
     *<p>
     * Feature is enabled by default, and is similar in behavior
     * to default prior to 1.7.
     *
     * @since 1.7
     */
    WRAP_EXCEPTIONS(true),

    /*
    /**********************************************************
     * Output life cycle features
    /**********************************************************
     */

     /**
      * Feature that determines whether <code>close</code> method of
      * serialized <b>root level</b> objects (ones for which <code>ObjectMapper</code>'s
      * writeValue() (or equivalent) method is called)
      * that implement {@link java.io.Closeable}
      * is called after serialization or not. If enabled, <b>close()</b> will
      * be called after serialization completes (whether succesfully, or
```

```
     * due to an error manifested by an exception being thrown). You can
     * think of this as sort of "finally" processing.
     *<p>
     * NOTE: only affects behavior with <b>root</b> objects, and not other
     * objects reachable from the root object. Put another way, only one
     * call will be made for each 'writeValue' call.
     *
     * @since 1.6 (see [JACKSON-282 for details])
     */
    CLOSE_CLOSEABLE(false),

    /**
     * Feature that determines whether <code>JsonGenerator.flush()</code> is
     * called after <code>writeValue()</code> method <b>that takes JsonParser
     * as an argument</b> completes or not (i.e. does NOT affect methods
     * that use other destinations); same for methods in {@link ObjectWriter}.
     * This usually makes sense; but there are cases where flushing
     * should not be forced: for example when underlying stream is
     * compressing and flush() causes compression state to be flushed
     * (which occurs with some compression codecs).
     *
     * @since 1.6 (see [JACKSON-401 for details])
     */
    FLUSH_AFTER_WRITE_VALUE(true),

    /*
    /**********************************************************
     * Datatype-specific serialization configuration
    /**********************************************************
     */

    /**
     * Feature that determines whether {@link java.util.Date}s
     * (and Date-based things like {@link java.util.Calendar}s) are to be
     * serialized as numeric timestamps (true; the default),
     * or as something else (usually textual representation).
     * If textual representation is used, the actual format is
     * one returned by a call to {@link #getDateFormat}.
     *<p>
     * Note: whether this feature affects handling of other date-related
     * types depend on handlers of those types.
     */
    WRITE_DATES_AS_TIMESTAMPS(true),

    /**
     * Feature that determines how type <code>char[]</code> is serialized:
     * when enabled, will be serialized as an explict JSON array (with
     * single-character Strings as values); when disabled, defaults to
```

```
 * serializing them as Strings (which is more compact).
 *
 * @since 1.6 (see [JACKSON-289 for details])
 */
WRITE_CHAR_ARRAYS_AS_JSON_ARRAYS(false),

/**
 * Feature that determines standard serialization mechanism used for
 * Enum values: if enabled, return value of <code>Enum.toString()</code>
 * is used; if disabled, return value of <code>Enum.name()</code> is used.
 * Since pre-1.6 method was to use Enum name, this is the default.
 *<p>
 * Note: this feature should usually have same value
 * as {@link DeserializationConfig.Feature#READ_ENUMS_USING_TO_STRING}.
 *<p>
 * For further details, check out [JACKSON-212]
 *
 * @since 1.6
 */
WRITE_ENUMS_USING_TO_STRING(false),

/**
 * Feature that determines whether Map entries with null values are
 * to be serialized (true) or not (false).
 *<p>
 * For further details, check out [JACKSON-314]
 *
 * @since 1.6
 */
WRITE_NULL_MAP_VALUES(true)

   ;

final boolean _defaultState;

/**
 * Method that calculates bit set (flags) of all features that
 * are enabled by default.
 */
public static int collectDefaults()
{
   int flags = 0;
   for (Feature f : values()) {
      if (f.enabledByDefault()) {
         flags |= f.getMask();
      }
   }
   return flags;
```

```
    }

    private Feature(boolean defaultState) {
        _defaultState = defaultState;
    }

    public boolean enabledByDefault() { return _defaultState; }

    public int getMask() { return (1 << ordinal()); }
}

/**
 * Bitfield (set of flags) of all Features that are enabled
 * by default.
 */
protected final static int DEFAULT_FEATURE_FLAGS = Feature.collectDefaults();

/*
/**********************************************************
/* Configuration settings
/**********************************************************
 */


/**
 * Introspector used to figure out Bean properties needed for bean serialization
 * and deserialization. Overridable so that it is possible to change low-level
 * details of introspection, like adding new annotation types.
 */
protected ClassIntrospector<? extends BeanDescription> _classIntrospector;

/**
 * Introspector used for accessing annotation value based configuration.
 */
protected AnnotationIntrospector _annotationIntrospector;

protected int _featureFlags = DEFAULT_FEATURE_FLAGS;

/**
 * Textual date format to use for serialization (if enabled by
 * {@link Feature#WRITE_DATES_AS_TIMESTAMPS} being set to false).
 * Defaults to a ISO-8601 compliant format used by
 * {@link StdDateFormat}.
 *<p>
 * Note that format object is <b>not to be used as is</b> by caller:
 * since date format objects are not thread-safe, caller has to
 * create a clone first.
 */
```

```
protected DateFormat _dateFormat = StdDateFormat.instance;

/**
 * Which Bean/Map properties are to be included in serialization?
 * Default settings is to include all regardless of value; can be
 * changed to only include non-null properties, or properties
 * with non-default values.
 *<p>
 * Defaults to null for backwards compatibility; if left as null,
 * will check
 * deprecated {@link Feature#WRITE_NULL_PROPERTIES}
 * to choose between {@link Inclusion#ALWAYS}
 * and {@link Inclusion#NON_NULL}.
 */
protected JsonSerialize.Inclusion _serializationInclusion = null;

/**
 * View to use for filtering out properties to serialize.
 * Null if none (will also be assigned null if <code>Object.class</code>
 * is defined), meaning that all properties are to be included.
 */
protected Class<?> _serializationView;

/**
 * Mapping that defines how to apply mix-in annotations: key is
 * the type to received additional annotations, and value is the
 * type that has annotations to "mix in".
 *<p>
 * Annotations associated with the value classes will be used to
 * override annotations of the key class, associated with the
 * same field or method. They can be further masked by sub-classes:
 * you can think of it as injecting annotations between the target
 * class and its sub-classes (or interfaces)
 *
 * @since 1.2
 */
protected HashMap<ClassKey,Class<?>> _mixInAnnotations;

/**
 * Flag used to detect when a copy if mix-in annotations is
 * needed: set when current copy is shared, cleared when a
 * fresh copy is maed
 *
 * @since 1.2
 */
protected boolean _mixInAnnotationsShared;

/**
```

```
     * Type information handler used for "untyped" values (ones declared
     * to have type <code>Object.class</code>)
     *
     * @since 1.5
     */
    protected final TypeResolverBuilder<?> _typer;

    /**
     * Object used for determining whether specific property elements
     * (method, constructors, fields) can be auto-detected based on
     * their visibility (access modifiers). Can be changed to allow
     * different minimum visibility levels for auto-detection. Note
     * that this is the global handler; individual types (classes)
     * can further override active checker used (using
     * {@link JsonAutoDetect} annotation)
     *
     * @since 1.5
     */
    protected VisibilityChecker<?> _visibilityChecker;

    /**
     * Registered concrete subtypes that can be used instead of (or
     * in addition to) ones declared using annotations.
     *
     * @since 1.6
     */
    protected SubtypeResolver _subtypeResolver;

    /**
     * Object used for resolving filter ids to filter instances.
     * Non-null if explicitly defined; null by default.
     *
     * @since 1.7
     */
    protected FilterProvider _filterProvider;

    /*
    /**********************************************************
    /* Life-cycle
    /**********************************************************
     */

    public SerializationConfig(ClassIntrospector<? extends BeanDescription> intr,
            AnnotationIntrospector annIntr, VisibilityChecker<?> vc,
            SubtypeResolver subtypeResolver)
    {
        _classIntrospector = intr;
        _annotationIntrospector = annIntr;
```

```
    _typer = null;
    _visibilityChecker = vc;
    _subtypeResolver = subtypeResolver;
    _filterProvider = null;
}

/**
 * @since 1.7
 */
protected SerializationConfig(SerializationConfig src,
        HashMap<ClassKey,Class<?>> mixins,
        TypeResolverBuilder<?> typer,
        VisibilityChecker<?> vc,
        SubtypeResolver subtypeResolver,
        FilterProvider filterProvider)
{
    _classIntrospector = src._classIntrospector;
    _annotationIntrospector = src._annotationIntrospector;
    _featureFlags = src._featureFlags;
    _dateFormat = src._dateFormat;
    _serializationInclusion = src._serializationInclusion;
    _serializationView = src._serializationView;
    _mixInAnnotations = mixins;
    _typer = typer;
    _visibilityChecker = vc;
    _subtypeResolver = subtypeResolver;
    _filterProvider = filterProvider;
}

/**
 * Copy constructor used for creating a new instance that is same as
 * this one, but with different <code>FilterProvider</code>.
 *
 * @since 1.7
 */
protected SerializationConfig(SerializationConfig src, FilterProvider filterProvider)
{
    _classIntrospector = src._classIntrospector;
    _annotationIntrospector = src._annotationIntrospector;
    _featureFlags = src._featureFlags;
    _dateFormat = src._dateFormat;
    _serializationInclusion = src._serializationInclusion;
    _serializationView = src._serializationView;
    _mixInAnnotations = src._mixInAnnotations;
    _typer = src._typer;
    _visibilityChecker = src._visibilityChecker;
    _subtypeResolver = src._subtypeResolver;
    _filterProvider = filterProvider;
```

```
}

public SerializationConfig withFilters(FilterProvider filterProvider) {
   return new SerializationConfig(this, filterProvider);
}

/**
 * SerializationConfig-specific version for constructing unshared
 * configuration object.
 *
 * @since 1.7
 */
public SerializationConfig createUnshared(TypeResolverBuilder<?> typer,
          VisibilityChecker<?> vc, SubtypeResolver subtypeResolver,
          FilterProvider filterProvider)
{
   HashMap<ClassKey,Class<?>> mixins = _mixInAnnotations;
   _mixInAnnotationsShared = true;
   return new SerializationConfig(this, mixins, typer, vc, subtypeResolver, filterProvider);
}

/*
/**********************************************************
/* MapperConfig implementation
/**********************************************************
 */

/**
 * Method that checks class annotations that the argument Object has,
 * and modifies settings of this configuration object accordingly,
 * similar to how those annotations would affect actual value classes
 * annotated with them, but with global scope. Note that not all
 * annotations have global significance, and thus only subset of
 * Jackson annotations will have any effect.
 *<p>
 * Serialization annotations that are known to have effect are:
 *<ul>
 * <li>{@link JsonWriteNullProperties}</li>
 * <li>{@link JsonAutoDetect}</li>
 * <li>{@link JsonSerialize#typing}</li>
 *</ul>
 *
 * @param cls Class of which class annotations to use
 *   for changing configuration settings
 */
//@Override
public void fromAnnotations(Class<?> cls)
{
```

```
    /* 10-Jul-2009, tatu: Should be able to just pass null as
     *    'MixInResolver'; no mix-ins set at this point
     * 29-Jul-2009, tatu: Also, we do NOT ignore annotations here, even
     *    if Feature.USE_ANNOTATIONS was disabled, since caller
     *    specifically requested annotations to be added with this call
     */
    AnnotatedClass ac = AnnotatedClass.construct(cls, _annotationIntrospector, null);
    _visibilityChecker = _annotationIntrospector.findAutoDetectVisibility(ac, _visibilityChecker);

    // How about writing null property values?
    JsonSerialize.Inclusion incl = _annotationIntrospector.findSerializationInclusion(ac, null);
    if (incl != _serializationInclusion) {
        setSerializationInclusion(incl);
 }

    JsonSerialize.Typing typing = _annotationIntrospector.findSerializationTyping(ac);
    if (typing != null) {
        set(Feature.USE_STATIC_TYPING, (typing == JsonSerialize.Typing.STATIC));
    }
}

/**
 * Method that is called to create a non-shared copy of the configuration
 * to be used for a serialization operation.
 * Note that if sub-classing
 * and sub-class has additional instance methods,
 * this method <b>must</b> be overridden to produce proper sub-class
 * instance.
 */
//@Override
public SerializationConfig createUnshared(TypeResolverBuilder<?> typer,
        VisibilityChecker<?> vc, SubtypeResolver subtypeResolver)
{
    HashMap<ClassKey,Class<?>> mixins = _mixInAnnotations;
    _mixInAnnotationsShared = true;
    return new SerializationConfig(this, mixins, typer, vc, subtypeResolver, _filterProvider);
}

//@Override
public void setIntrospector(ClassIntrospector<? extends BeanDescription> i) {
    _classIntrospector = i;
}

/**
 * Method for getting {@link AnnotationIntrospector} configured
 * to introspect annotation values used for configuration.
 */
//@Override
```

```
public AnnotationIntrospector getAnnotationIntrospector()
{
   /* 29-Jul-2009, tatu: it's now possible to disable use of
    *   annotations; can be done using "no-op" introspector
    */
   if (isEnabled(Feature.USE_ANNOTATIONS)) {
      return _annotationIntrospector;
   }
   return AnnotationIntrospector.nopInstance();
}

//@Override
public void setAnnotationIntrospector(AnnotationIntrospector ai) {
   _annotationIntrospector = ai;
}

//@Override
public void insertAnnotationIntrospector(AnnotationIntrospector introspector)
{
   _annotationIntrospector = AnnotationIntrospector.Pair.create(introspector, _annotationIntrospector);
}

//@Override
public void appendAnnotationIntrospector(AnnotationIntrospector introspector)
{
   _annotationIntrospector = AnnotationIntrospector.Pair.create(_annotationIntrospector, introspector);
}

/**
 * Method to use for defining mix-in annotations to use for augmenting
 * annotations that serializable classes have.
 * Mixing in is done when introspecting class annotations and properties.
 * Map passed contains keys that are target classes (ones to augment
 * with new annotation overrides), and values that are source classes
 * (have annotations to use for augmentation).
 * Annotations from source classes (and their supertypes)
 * will <b>override</b>
 * annotations that target classes (and their super-types) have.
 *<p>
 * Note: a copy of argument Map is created; the original Map is
 * not modified or retained by this config object.
 *
 * @since 1.2
 */
//@Override
public void setMixInAnnotations(Map<Class<?>, Class<?>> sourceMixins)
{
   HashMap<ClassKey,Class<?>> mixins = null;
```

```
    if (sourceMixins != null && sourceMixins.size() > 0) {
        mixins = new HashMap<ClassKey,Class<?>>(sourceMixins.size());
        for (Map.Entry<Class<?>,Class<?>> en : sourceMixins.entrySet()) {
            mixins.put(new ClassKey(en.getKey()), en.getValue());
        }
    }
    _mixInAnnotationsShared = false;
    _mixInAnnotations = mixins;
}

//@Override
public void addMixInAnnotations(Class<?> target, Class<?> mixinSource)
{
    if (_mixInAnnotations == null || _mixInAnnotationsShared) {
        _mixInAnnotationsShared = false;
        _mixInAnnotations = new HashMap<ClassKey,Class<?>>();
    }
    _mixInAnnotations.put(new ClassKey(target), mixinSource);
}

/**
 * @since 1.2
 */
//@Override
public Class<?> findMixInClassFor(Class<?> cls) {
    return (_mixInAnnotations == null) ? null : _mixInAnnotations.get(new ClassKey(cls));
}

//@Override
public DateFormat getDateFormat() { return _dateFormat; }

/**
 * Method that will set the specific date format to use for
 * serializing Dates (and Calendars); or if null passed, simply
 * disable textual serialization and use timestamp.
 * In addition to setting format, will also enable/disable feature
 * {@link Feature#WRITE_DATES_AS_TIMESTAMPS}: enable, if argument
 * is null; disable if non-null.
 */
//@Override
public void setDateFormat(DateFormat df) {
    _dateFormat = df;
    // Also: enable/disable usage of
    set(Feature.WRITE_DATES_AS_TIMESTAMPS, (df == null));
}

//@Override
public TypeResolverBuilder<?> getDefaultTyper(JavaType baseType) {
```

```
    return _typer;
}

//@Override
public VisibilityChecker<?> getDefaultVisibilityChecker() {
    return _visibilityChecker;
}

/**
 * Accessor for getting bean description that only contains class
 * annotations: useful if no getter/setter/creator information is needed.
 *<p>
 * Note: part of {@link MapperConfig} since 1.7
 */
@SuppressWarnings("unchecked")
public <T extends BeanDescription> T introspectClassAnnotations(Class<?> cls) {
    return (T) _classIntrospector.forClassAnnotations(this, cls, this);
}

/**
 * Accessor for getting bean description that only contains immediate class
 * annotations: ones from the class, and its direct mix-in, if any, but
 * not from super types.
 *<p>
 * Note: part of {@link MapperConfig} since 1.7
 */
@SuppressWarnings("unchecked")
public <T extends BeanDescription> T introspectDirectClassAnnotations(Class<?> cls) {
    return (T) _classIntrospector.forDirectClassAnnotations(this, cls, this);
}

/*
/**********************************************************
/* Configuration: on/off features
/**********************************************************
 */

/**
 * Method for enabling specified feature.
 */
public void enable(Feature f) {
    _featureFlags |= f.getMask();
}

/**
 * Method for disabling specified feature.
 */
public void disable(Feature f) {
```

```
    _featureFlags &= ~f.getMask();
}

/**
 * Method for enabling or disabling specified feature.
 */
public void set(Feature f, boolean state)
{
    if (state) {
        enable(f);
    } else {
        disable(f);
    }
}

//protected int getFeatures() { return _generatorFeatures; }

/**
 * Method for checking whether given feature is enabled or not
 */
public final boolean isEnabled(Feature f) {
    return (_featureFlags & f.getMask()) != 0;
}

/*
/**********************************************************
/* Introspection methods
/**********************************************************
 */

/**
 * Method that will introspect full bean properties for the purpose
 * of building a bean serializer
 */
@SuppressWarnings("unchecked")
public <T extends BeanDescription> T introspect(JavaType type) {
    return (T) _classIntrospector.forSerialization(this, type, this);
}

/*
/**********************************************************
/* Polymorphic type handling configuration
/**********************************************************
 */

/**
 * @since 1.6
 */
```

```
public SubtypeResolver getSubtypeResolver() {
   if (_subtypeResolver == null) {
      _subtypeResolver = new StdSubtypeResolver();
   }
   return _subtypeResolver;
}

/**
 * @since 1.6
 */
public void setSubtypeResolver(SubtypeResolver r) {
   _subtypeResolver = r;
}

/*
/**********************************************************
/* Configuration: other
/**********************************************************
 */

/**
 * Method for checking which serialization view is being used,
 * if any; null if none.
 *
 * @since 1.4
 */
public Class<?> getSerializationView() { return _serializationView; }

public JsonSerialize.Inclusion getSerializationInclusion()
{
   if (_serializationInclusion != null) {
      return _serializationInclusion;
   }
   return isEnabled(Feature.WRITE_NULL_PROPERTIES) ?
      JsonSerialize.Inclusion.ALWAYS : JsonSerialize.Inclusion.NON_NULL;
}

/**
 * Method that will define global setting of which
 * bean/map properties are to be included in serialization.
 * Can be overridden by class annotations (overriding
 * settings to use for instances of that class) and
 * method/field annotations (overriding settings for the value
 * bean for that getter method or field)
 */
public void setSerializationInclusion(JsonSerialize.Inclusion props)
{
   _serializationInclusion = props;
```

```
        // And for some level of backwards compatibility, also...
        if (props == JsonSerialize.Inclusion.NON_NULL) {
            disable(Feature.WRITE_NULL_PROPERTIES);
        } else {
            enable(Feature.WRITE_NULL_PROPERTIES);
        }
    }

    /**
     * Method for checking which serialization view is being used,
     * if any; null if none.
     *
     * @since 1.4
     */
    public void setSerializationView(Class<?> view)
    {
        _serializationView = view;
    }

    /**
     * Method for getting provider used for locating filters given
     * id (which is usually provided with filter annotations).
     * Will be null if no provided was set for {@link ObjectWriter}
     * (or if serialization directly called from {@link ObjectMapper})
     *
     * @since 1.7
     */
    public FilterProvider getFilterProvider() {
        return _filterProvider;
    }

    /*
    /**********************************************************
    /* Debug support
    /**********************************************************
     */

    @Override public String toString()
    {
        return "[SerializationConfig: flags=0x"+Integer.toHexString(_featureFlags)+"]";
    }
}
package org.codehaus.jackson.map;

import org.codehaus.jackson.type.JavaType;

/**
* Defines interface for resolvers that can resolve abstract types into concrete
```

```
* ones; either by using static mappings, or possibly by materializing
* implementations dynamically.
*
* @since 1.6
*/
public abstract class AbstractTypeResolver
{
  /**
   * Method called to try to resolve an abstract type into
   * concrete type, usually for purposes of deserializing.
   *
   * @param config Deserialization configuration in use
   * @param type Abstract type (with generic type parameters if any)
   *   to resolve
   *
   * @return Resolved concrete type (which should retain generic
   *    type parameters of input type, if any), if resolution succeeds;
   *    null if resolver does not know how to resolve type
   */
  public abstract JavaType resolveAbstractType(DeserializationConfig config, JavaType type);
}
package org.codehaus.jackson.map;

import java.io.IOException;

import org.codehaus.jackson.*;

/**
* Abstract class that defines API used by {@link ObjectMapper} (and
* other chained {@link JsonSerializer}s too) to serialize Objects of
* arbitrary types into JSON, using provided {@link JsonGenerator}.
*/
public abstract class JsonSerializer<T>
{
  /**
   * Method that can be called to ask implementation to serialize
   * values of type this serializer handles.
   *
   * @param value Value to serialize; can <b>not</b> be null.
   * @param jgen Generator used to output resulting Json content
   * @param provider Provider that can be used to get serializers for
   *   serializing Objects value contains, if any.
   */
  public abstract void serialize(T value, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonProcessingException;

  /**
   * Method that can be called to ask implementation to serialize
```

```
 * values of type this serializer handles, using specified type serializer
 * for embedding necessary type information.
 *<p>
 * Default implementation will ignore serialization of type information,
 * and just calls {@link #serialize}: serializers that can embed
 * type information should override this to implement actual handling.
 * Most common such handling is done by something like:
 *<pre>
 *  // note: method to call depends on whether this type is serialized as JSON scalar, object or Array!
 *  typeSer.writeTypePrefixForScalar(value, jgen);
 *  serialize(value, jgen, provider);
 *  typeSer.writeTypeSuffixForScalar(value, jgen);
 *</pre>
 *
 * @param value Value to serialize; can <b>not</b> be null.
 * @param jgen Generator used to output resulting Json content
 * @param provider Provider that can be used to get serializers for
 *   serializing Objects value contains, if any.
 * @param typeSer Type serializer to use for including type information
 *
 * @since 1.5
 */
public void serializeWithType(T value, JsonGenerator jgen, SerializerProvider provider,
        TypeSerializer typeSer)
   throws IOException, JsonProcessingException
{
   serialize(value, jgen, provider);
}

/*
/**********************************************************
* Introspection methods needed for type handling
/**********************************************************
 */

/**
 * Method for accessing type of Objects this serializer can handle.
 * Note that this information is not guaranteed to be exact -- it
 * may be a more generic (super-type) -- but it should not be
 * incorrect (return a non-related type).
 *<p>
 * Default implementation will return null, which essentially means
 * same as returning <code>Object.class</code> would; that is, that
 * nothing is known about handled type.
 *<p>
 */
public Class<T> handledType() { return null; }
```

```
  /*
  /**********************************************************
  /* Helper class(es)
  /**********************************************************
   */

  /**
   * This marker class is only to be used with annotations, to
   * indicate that <b>no serializer is configured</b>.
   *<p>
   * Specifically, this class is to be used as the marker for
   * annotation {@link org.codehaus.jackson.map.annotate.JsonSerialize}.
   */
  public abstract static class None
      extends JsonSerializer<Object> { }
}
package org.codehaus.jackson.map;

/**
* Wrapper used when interface does not allow throwing a checked
* {@link JsonMappingException}
*/
@SuppressWarnings("serial")
public class RuntimeJsonMappingException extends RuntimeException
{
  public RuntimeJsonMappingException(JsonMappingException cause) {
      super(cause);
  }

  public RuntimeJsonMappingException(String message) {
      super(message);
  }

  public RuntimeJsonMappingException(String message, JsonMappingException cause) {
      super(message, cause);
  }
}
package org.codehaus.jackson.map.exc;

import org.codehaus.jackson.JsonLocation;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.map.JsonMappingException;

/**
* Specialized {@link JsonMappingException} sub-class specifically used
* to indicate problems due to encountering a JSON property that could
* not be mapped to an Object property (via getter, constructor argument
* or field).
```

```
*
* @since 1.6
*/
public class UnrecognizedPropertyException
  extends JsonMappingException
{
  private static final long serialVersionUID = 1L;

  /**
   * Class that does not contain mapping for the unrecognized property.
   */
  protected final Class<?> _referringClass;

  /**
   *<p>
   * Note: redundant information since it is also included in the
   * reference path.
   */
  protected final String _unrecognizedPropertyName;

  public UnrecognizedPropertyException(String msg, JsonLocation loc,
        Class<?> referringClass, String propName)
  {

    super(msg, loc);
    _referringClass = referringClass;
    _unrecognizedPropertyName = propName;
  }

  public static UnrecognizedPropertyException from(JsonParser jp, Object fromObjectOrClass, String
propertyName)
  {
    if (fromObjectOrClass == null) {
      throw new IllegalArgumentException();
    }
    Class<?> ref;
    if (fromObjectOrClass instanceof Class<?>) {
      ref = (Class<?>) fromObjectOrClass;
    } else {
      ref = fromObjectOrClass.getClass();
    }
    String msg = "Unrecognized field \""+propertyName+"\" (Class "+ref.getName()+"), not marked as ignorable";
    UnrecognizedPropertyException e = new UnrecognizedPropertyException(msg, jp.getCurrentLocation(), ref,
propertyName);
    // but let's also ensure path includes this last (missing) segment
    e.prependPath(fromObjectOrClass, propertyName);
    return e;
  }
```

```
  /**
   * Method for accessing type (class) that is missing definition to allow
   * binding of the unrecognized property.
   */
  public Class<?> getReferringClass() {
     return _referringClass;
  }

  /**
   * Convenience method for accessing logical property name that could
   * not be mapped. Note that it is the last path reference in the
   * underlying path.
   */
  public String getUnrecognizedPropertyName() {
     return _unrecognizedPropertyName;
  }
}
/**
* Package that contains concrete implementations of
* {@link org.codehaus.jackson.type.JavaType}, as
* well as the factory ({@link org.codehaus.jackson.map.type.TypeFactory}) for
* constructing instances from various input data types
* (like {@link java.lang.Class}, {@link java.lang.reflect.Type})
* and programmatically (for structured types, arrays,
* {@link java.util.List}s and {@link java.util.Map}s).
*/
package org.codehaus.jackson.map.type;
package org.codehaus.jackson.map.type;

import java.lang.reflect.Array;

import org.codehaus.jackson.type.JavaType;

/**
* Array types represent Java arrays, both primitive and object valued.
* Further, Object-valued arrays can have element type of any other
* legal {@link JavaType}.
*/
public final class ArrayType
  extends TypeBase
{
  /**
   * Type of elements in the array.
   */
  final JavaType _componentType;

  /**
```

```
     * We will also keep track of shareable instance of empty array,
     * since it usually needs to be constructed any way; and because
     * it is essentially immutable and thus can be shared.
     */
    final Object _emptyArray;

    private ArrayType(JavaType componentType, Object emptyInstance)
    {
        super(emptyInstance.getClass(), componentType.hashCode());
        _componentType = componentType;
        _emptyArray = emptyInstance;
    }

    public static ArrayType construct(JavaType componentType)
    {
        /* This is bit messy: there is apparently no other way to
         * reconstruct actual concrete/raw array class from component
         * type, than to construct an instance, get class (same is
         * true for GenericArracyType as well; hence we won't bother
         * passing that in).
         */
        Object emptyInstance = Array.newInstance(componentType.getRawClass(), 0);
        return new ArrayType(componentType, emptyInstance);
    }

    // Since 1.7:
    @Override
    public ArrayType withTypeHandler(Object h)
    {
        ArrayType newInstance = new ArrayType(_componentType, _emptyArray);
        newInstance._typeHandler = h;
        return newInstance;
    }

    // Since 1.7:
    @Override
    public ArrayType withContentTypeHandler(Object h)
    {
        return new ArrayType(_componentType.withTypeHandler(h), _emptyArray);
    }

    @Override
    protected String buildCanonicalName() {
        return _class.getName();
    }

    /*
    /**********************************************************
```

```
    /* Methods for narrowing conversions
    /**********************************************************
     */

    /**
     * Handling of narrowing conversions for arrays is trickier: for now,
     * it is not even allowed.
     */
    @Override
    protected JavaType _narrow(Class<?> subclass)
    {
        /* Ok: need a bit of indirection here. First, must replace component
         * type (and check that it is compatible), then re-construct.
         */
        if (!subclass.isArray()) { // sanity check, should never occur
            throw new IllegalArgumentException("Incompatible narrowing operation: trying to narrow "+toString()+" to
class "+subclass.getName());
        }
        /* Hmmh. This is an awkward back reference... but seems like the
         * only simple way to do it.
         */
        Class<?> newCompClass = subclass.getComponentType();
        JavaType newCompType = TypeFactory.type(newCompClass);
        return construct(newCompType);
    }

    /**
     * For array types, both main type and content type can be modified;
     * but ultimately they are interchangeable.
     */
    @Override
    public JavaType narrowContentsBy(Class<?> contentClass)
    {
        // Can do a quick check first:
        if (contentClass == _componentType.getRawClass()) {
            return this;
        }
        JavaType newComponentType = _componentType.narrowBy(contentClass);
        return construct(newComponentType).copyHandlers(this);
    }

    /*
    /**********************************************************
    /* Overridden methods
    /**********************************************************
     */

    @Override
```

```
public boolean isArrayType() { return true; }

/**
 * For some odd reason, modifiers for array classes would
 * claim they are abstract types. Not so, at least for our
 * purposes.
 */
@Override
public boolean isAbstract() { return false; }

/**
 * For some odd reason, modifiers for array classes would
 * claim they are abstract types. Not so, at least for our
 * purposes.
 */
@Override
public boolean isConcrete() { return true; }

@Override
public boolean hasGenericTypes() {
   // arrays are not parameterized, but element type may be:
   return _componentType.hasGenericTypes();
}

/**
 * Not sure what symbolic name is used internally, if any;
 * let's follow naming of Collection types here.
 * Should not really matter since array types have no
 * super types.
 */
@Override
public String containedTypeName(int index) {
   if (index == 0) return "E";
   return null;
}

/*
/**********************************************************
/* Public API
/**********************************************************
 */

@Override
public boolean isContainerType() { return true; }

@Override
public JavaType getContentType() { return  _componentType; }
```

```
  @Override
  public int containedTypeCount() { return 1; }
  @Override
  public JavaType containedType(int index) {
        return (index == 0) ? _componentType : null;
  }

  @Override
  public StringBuilder getGenericSignature(StringBuilder sb) {
     sb.append('[');
     return _componentType.getGenericSignature(sb);
  }

  @Override
  public StringBuilder getErasedSignature(StringBuilder sb) {
     sb.append('[');
     return _componentType.getErasedSignature(sb);
  }

  /*
  /**********************************************************
  /* Standard methods
  /**********************************************************
   */

  @Override
  public String toString()
  {
     return "[array type, component type: "+_componentType+"]";
  }

  @Override
  public boolean equals(Object o)
  {
     if (o == this) return true;
     if (o == null) return false;
     if (o.getClass() != getClass()) return false;

     ArrayType other = (ArrayType) o;
     return _componentType.equals(other._componentType);
  }
}
package org.codehaus.jackson.map.type;

import java.lang.reflect.*;

/**
* Simple replacement for {@link java.lang.Class} (and/or various Type subtypes)
```

```
* that is used as part of single-path extends/implements chain to express
* specific relationship between one subtype and one supertype. This is needed
* for resolving type parameters. Instances are doubly-linked so that chain
* can be traversed in both directions
*
* @since 1.6
*/
public class HierarchicType
{
  /**
   * Type which will be either plain {@link java.lang.Class} or
   * {@link java.lang.reflect.ParameterizedType}.
   */
  protected final Type _actualType;

  protected final Class<?> _rawClass;

  protected final ParameterizedType _genericType;

  protected HierarchicType _superType;

  protected HierarchicType _subType;

  public HierarchicType(Type type)
  {
     this._actualType = type;
     if (type instanceof Class<?>) {
        _rawClass = (Class<?>) type;
        _genericType = null;
     } else if (type instanceof ParameterizedType) {
        _genericType = (ParameterizedType) type;
        _rawClass = (Class<?>) _genericType.getRawType();
     } else { // should never happen... can't extend GenericArrayType?
        throw new IllegalArgumentException("Type "+type.getClass().getName()+" can not be used to construct
HierarchicType");
     }
  }

  public void setSuperType(HierarchicType sup) { _superType = sup; }
  public HierarchicType getSuperType() { return _superType; }
  public void setSubType(HierarchicType sub) { _subType = sub; }
  public HierarchicType getSubType() { return _subType; }

  public boolean isGeneric() { return _genericType != null; }
  public ParameterizedType asGeneric() { return _genericType; }

  public Class<?> getRawClass() { return _rawClass; }
```

```
  @Override
  public String toString() {
     if (_genericType != null) {
        return _genericType.toString();
     }
     return _rawClass.getName();
  }

}
package org.codehaus.jackson.map.type;

import java.util.*;
import java.lang.reflect.*;

import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.type.TypeReference;

/**
* Class used for constracting concrete {@link JavaType} instances,
* given various inputs.
*<p>
* Typical usage patterns is to statically import factory methods
* of this class, to allow convenient instantiation of structured
* types, especially {@link Collection} and {@link Map} types
* to represent generic types. For example
*<pre>
* mapType(String.class, Integer.class)
*</pre>
* to represent
*<pre>
*  Map<String,Integer>
*</pre>
* This is an alternative to using {@link TypeReference} that would
* be something like
*<pre>
*  new TypeReference<Map<String,Integer>>() { }
*</pre>
*/
public class TypeFactory
{
  public final static TypeFactory instance = new TypeFactory();

  private final static JavaType[] NO_TYPES = new JavaType[0];

  protected final TypeParser _parser;

  /*
  /**********************************************************
```

```
/* Life-cycle
/**********************************************************
 */

private TypeFactory() {
   _parser = new TypeParser(this);
}

/*
/**********************************************************
/* Public factory methods
/**********************************************************
 */

/**
 * Factory method for constructing {@link JavaType} from given
 * "raw" type; which may be anything from simple {@link Class}
 * to full generic type.
 *
 * @since 1.3
 */
public static JavaType type(Type t)
{
   return instance._fromType(t, null);
}

/**
 * Factory method that can use given context to resolve
 * named generic types.
 *
 * @param context Class used for resolving generic types; for example,
 *    for bean properties the actual bean class (not necessarily class
 *    that contains method or field, may be a sub-class thereof)
 */
public static JavaType type(Type type, Class<?> context)
{
   return type(type, new TypeBindings(context));
}

/**
 * @since 1.7.0
 */
public static JavaType type(Type type, JavaType context)
{
   return type(type, new TypeBindings(context));
}

public static JavaType type(Type type, TypeBindings bindings)
```

```
{
   return instance._fromType(type, bindings);
}

/**
 * Factory method that can be used if the full generic type has
 * been passed using {@link TypeReference}. This only needs to be
 * done if the root type to bind to is generic; but if so,
 * it must be done to get proper typing.
 */
public static JavaType type(TypeReference<?> ref)
{
   return type(ref.getType());
}

/**
 * Convenience factory method for constructing {@link JavaType} that
 * represent array that contains elements
 * of specified type.
 *
 * @since 1.3
 */
public static JavaType arrayType(Class<?> elementType)
{
   return arrayType(type(elementType));
}

/**
 * Convenience factory method for constructing {@link JavaType} that
 * represent array that contains elements
 * of specified type.
 *
 * @since 1.3
 */
public static JavaType arrayType(JavaType elementType)
{
   return ArrayType.construct(elementType);
}

/**
 * Convenience factory method for constructing {@link JavaType} that
 * represent Collection of specified type and contains elements
 * of specified type
 *
 * @since 1.3
 */
@SuppressWarnings("unchecked")
public static JavaType collectionType(Class<? extends Collection> collectionType, Class<?> elementType)
```

```
{
   return collectionType(collectionType, type(elementType));
}

/**
 * Convenience factory method for constructing {@link JavaType} that
 * represent Collection of specified type and contains elements
 * of specified type
 *
 * @since 1.3
 */
@SuppressWarnings("unchecked")
public static JavaType collectionType(Class<? extends Collection> collectionType, JavaType elementType)
{
   return CollectionType.construct(collectionType, elementType);
}

/**
 * Convenience factory method for constructing {@link JavaType} that
 * represent Map of specified type and contains elements
 * of specified type
 *
 * @since 1.3
 */
@SuppressWarnings("unchecked")
public static JavaType mapType(Class<? extends Map> mapType, Class<?> keyType, Class<?> valueType)
{
   return mapType(mapType, type(keyType), type(valueType));
}

/**
 * Convenience factory method for constructing {@link JavaType} that
 * represent Map of specified type and contains elements
 * of specified type
 *
 * @since 1.3
 */
@SuppressWarnings("unchecked")
public static JavaType mapType(Class<? extends Map> mapType, JavaType keyType, JavaType valueType)
{
   return MapType.construct(mapType, keyType, valueType);
}

/**
 * Factory method for constructing {@link JavaType} that
 * represents a parameterized type. For example, to represent
 * type <code>Iterator<String></code>, you could
 * call
```

```
     *<pre>
     *  TypeFactory.parametricType(Iterator.class, String.class)
     *</pre>
     *
     * @since 1.5
     */
    public static JavaType parametricType(Class<?> parametrized, Class<?>... parameterClasses)
    {
        int len = parameterClasses.length;
        JavaType[] pt = new JavaType[len];
        for (int i = 0; i < len; ++i) {
            pt[i] = instance._fromClass(parameterClasses[i], null);
        }
        return parametricType(parametrized, pt);
    }

    /**
     * Factory method for constructing {@link JavaType} that
     * represents a parameterized type. For example, to represent
     * type <code>List<Set<Integer>></code>, you could
     * call
     *<pre>
     *  JavaType inner = TypeFactory.parametricType(Set.class, Integer.class);
     *  TypeFactory.parametricType(List.class, inner);
     *</pre>
     *
     * @since 1.5
     */
    public static JavaType parametricType(Class<?> parametrized, JavaType... parameterTypes)
    {
        // Need to check kind of class we are dealing with...
        if (parametrized.isArray()) {
            // 19-Jan-2010, tatus: should we support multi-dimensional arrays directly?
            if (parameterTypes.length != 1) {
                throw new IllegalArgumentException("Need exactly 1 parameter type for arrays
("+parametrized.getName()+")");
            }
            return ArrayType.construct(parameterTypes[0]);
        }
        if (Map.class.isAssignableFrom(parametrized)) {
            if (parameterTypes.length != 2) {
                throw new IllegalArgumentException("Need exactly 2 parameter types for Map types
("+parametrized.getName()+")");
            }
            return MapType.construct(parametrized, parameterTypes[0], parameterTypes[1]);
        }
        if (Collection.class.isAssignableFrom(parametrized)) {
            if (parameterTypes.length != 1) {
```

```
            throw new IllegalArgumentException("Need exactly 1 parameter type for Collection types
("+parametrized.getName()+")");
        }
        return CollectionType.construct(parametrized, parameterTypes[0]);
    }
    return _constructSimple(parametrized, parameterTypes);
  }

  /**
   * Factory method for constructing a {@link JavaType} out of its canonical
   * representation (see {@link JavaType#toCanonical()}).
   *
   * @param canonical Canonical string representation of a type
   *
   * @throws IllegalArgumentException If canonical representation is malformed,
   *   or class that type represents (including its generic parameters) is
   *   not found
   *
   * @since 1.5
   */
  public static JavaType fromCanonical(String canonical)
    throws IllegalArgumentException
  {
    return instance._parser.parse(canonical);
  }

  /*
  /**********************************************************
  /* Type conversions
  /**********************************************************
   */

  /**
   * Method that tries to create specialized type given base type, and
   * a sub-class thereof (which is assumed to use same parametrization
   * as supertype). Similar to calling {@link JavaType#narrowBy(Class)},
   * but can change underlying {@link JavaType} (from simple to Map, for
   * example), unliked <code>narrowBy</code> which assumes same logical
   * type.
   */
  public static JavaType specialize(JavaType baseType, Class<?> subclass)
  {
    // Currently only SimpleType instances can become something else
   if (baseType instanceof SimpleType) {
        // and only if subclass is an array, Collection or Map
        if (subclass.isArray()
            || Map.class.isAssignableFrom(subclass)
            || Collection.class.isAssignableFrom(subclass)) {
```

```
            // need to assert type compatibility...
            if (!baseType.getRawClass().isAssignableFrom(subclass)) {
                throw new IllegalArgumentException("Class "+subclass.getClass().getName()+" not subtype of
"+baseType);
            }
            // this _should_ work, right?
            JavaType subtype = instance._fromClass(subclass, new TypeBindings(baseType.getRawClass()));
            // one more thing: handlers to copy?
            Object h = baseType.getValueHandler();
            if (h != null) {
                subtype.setValueHandler(h);
            }
            h = baseType.getTypeHandler();
            if (h != null) {
                subtype = subtype.withTypeHandler(h);
            }
            return subtype;
        }
    }
    // otherwise regular narrowing should work just fine
    return baseType.narrowBy(subclass);
}

/**
 * Method that can be used if it is known for sure that given type
 * is not a structured type (array, Map, Collection).
 * NOTE: use of this method is discouraged due to its potential
 * non-safety; in most cases you should just use basic
 * {@link #type(Type)} instead.
 *
 * @since 1.6
 */
public static JavaType fastSimpleType(Class<?> cls)
{
    return new SimpleType(cls, null, null);
}

/**
 * Method that is to figure out actual type parameters that given
 * class binds to generic types defined by given (generic)
 * interface or class.
 * This could mean, for example, trying to figure out
 * key and value types for Map implementations.
 *
 * @since 1.6
 */
public static JavaType[] findParameterTypes(Class<?> clz, Class<?> expType)
{
```

```
    return findParameterTypes(clz, expType, new TypeBindings(clz));
}

public static JavaType[] findParameterTypes(Class<?> clz, Class<?> expType, TypeBindings bindings)
{
    // First: find full inheritance chain
    HierarchicType subType = _findSuperTypeChain(clz, expType);
    // Caller is supposed to ensure this never happens, so:
    if (subType == null) {
        throw new IllegalArgumentException("Class "+clz.getName()+" is not a subtype of "+expType.getName());
    }
    // Ok and then go to the ultimate super-type:
    HierarchicType superType = subType;
    while (superType.getSuperType() != null) {
        superType = superType.getSuperType();
        Class<?> raw = superType.getRawClass();
        TypeBindings newBindings = new TypeBindings(raw);
        if (superType.isGeneric()) { // got bindings, need to resolve
            ParameterizedType pt = superType.asGeneric();
            Type[] actualTypes = pt.getActualTypeArguments();
            TypeVariable<?>[] vars = raw.getTypeParameters();
            int len = actualTypes.length;
            for (int i = 0; i < len; ++i) {
                String name = vars[i].getName();
                JavaType type = instance._fromType(actualTypes[i], bindings);
                newBindings.addBinding(name, type);
            }
        }
        bindings = newBindings;
    }

    // which ought to be generic (if not, it's raw type)
    if (!superType.isGeneric()) {
        return null;
    }
    return bindings.typesAsArray();
}

/**
 * Method that is to figure out actual type parameters that given
 * class binds to generic types defined by given (generic)
 * interface or class.
 * This could mean, for example, trying to figure out
 * key and value types for Map implementations.
 *
 * @param type Sub-type (leaf type) that implements <code>expType</code>
 *
 * @since 1.6
```

```
     */
    public static JavaType[] findParameterTypes(JavaType type, Class<?> expType)
    {
        /* Tricky part here is that some JavaType instances have been constructed
         * from generic type (usually via TypeReference); and in those case
         * types have been resolved. Alternative is that the leaf type is type-erased
         * class, in which case this has not been done.
         * For now simplest way to handle this is to split processing in two: latter
         * case actually fully works; and former mostly works. In future may need to
         * rewrite former part, which requires changes to JavaType as well.
         */
        Class<?> raw = type.getRawClass();
        if (raw == expType) {
            // Direct type info; good since we can return it as is
            int count = type.containedTypeCount();
            if (count == 0) return null;
            JavaType[] result = new JavaType[count];
            for (int i = 0; i < count; ++i) {
                result[i] = type.containedType(i);
            }
            return result;
        }
        /* Otherwise need to go through type-erased class. This may miss cases where
         * we get generic type; ideally JavaType/SimpleType would retain information
         * about generic declaration at main level... but let's worry about that
         * if/when there are problems; current handling is an improvement over earlier
         * code.
         */
        return findParameterTypes(raw, expType, new TypeBindings(type));
    }

    /*
    /**********************************************************
    /* Legacy methods
    /**********************************************************
     */

    /**
     * Factory method that can be used if only type information
     * available is of type {@link Class}. This means that there
     * will not be generic type information due to type erasure,
     * but at least it will be possible to recognize array
     * types and non-typed container types.
     * And for other types (primitives/wrappers, beans), this
     * is all that is needed.
     *
     * @deprecated Use {@link #type(Type)} instead
     */
```

```
@Deprecated
public static JavaType fromClass(Class<?> clz)
{
   return instance._fromClass(clz, null);
}

/**
 * Factory method that can be used if the full generic type has
 * been passed using {@link TypeReference}. This only needs to be
 * done if the root type to bind to is generic; but if so,
 * it must be done to get proper typing.
 *
 * @deprecated Use {@link #type(Type)} instead
 */
@Deprecated
public static JavaType fromTypeReference(TypeReference<?> ref)
{
   return type(ref.getType());
}

/**
 * Factory method that can be used if type information is passed
 * as Java typing returned from <code>getGenericXxx</code> methods
 * (usually for a return or argument type).
 *
 * @deprecated Use {@link #type(Type)} instead
 */
@Deprecated
public static JavaType fromType(Type type)
{
   return instance._fromType(type, null);
}

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

/**
 * @param context Mapping of formal parameter declarations (for generic
 *   types) into actual types
 */
protected JavaType _fromClass(Class<?> clz, TypeBindings context)
{
   // First: do we have an array type?
   if (clz.isArray()) {
      return ArrayType.construct(_fromType(clz.getComponentType(), null));
```

```
    }
    /* Also: although enums can also be fully resolved, there's little
     * point in doing so (T extends Enum<T>) etc.
     */
    if (clz.isEnum()) {
        return new SimpleType(clz);
    }
    /* Maps and Collections aren't quite as hot; problem is, due
     * to type erasure we often do not know typing and can only assume
     * base Object.
     */
    if (Map.class.isAssignableFrom(clz)) {
        return _mapType(clz);
    }
    if (Collection.class.isAssignableFrom(clz)) {
        return _collectionType(clz);
    }
    return new SimpleType(clz);
}

/**
 * Method used by {@link TypeParser} when generics-aware version
 * is constructed.
 */
protected JavaType _fromParameterizedClass(Class<?> clz, List<JavaType> paramTypes)
{
    if (clz.isArray()) { // ignore generics (should never have any)
        return ArrayType.construct(_fromType(clz.getComponentType(), null));
    }
    if (clz.isEnum()) { // ditto for enums
        return new SimpleType(clz);
    }
    if (Map.class.isAssignableFrom(clz)) {
        // First: if we do have param types, use them
        JavaType keyType, contentType;
        if (paramTypes.size() > 0) {
            keyType = paramTypes.get(0);
            contentType = (paramTypes.size() >= 2) ?
                    paramTypes.get(1) : _unknownType();
            return MapType.construct(clz, keyType, contentType);
        }
        return _mapType(clz);
    }
    if (Collection.class.isAssignableFrom(clz)) {
        if (paramTypes.size() >= 1) {
            return CollectionType.construct(clz, paramTypes.get(0));
        }
        return _collectionType(clz);
```

```
    }
    if (paramTypes.size() == 0) {
        return new SimpleType(clz);
    }
    JavaType[] pt = paramTypes.toArray(new JavaType[paramTypes.size()]);
    return _constructSimple(clz, pt);
}

/**
 * Factory method that can be used if type information is passed
 * as Java typing returned from <code>getGenericXxx</code> methods
 * (usually for a return or argument type).
 */
public JavaType _fromType(Type type, TypeBindings context)
{
    // simple class?
    if (type instanceof Class<?>) {
        Class<?> cls = (Class<?>) type;
        /* 24-Mar-2010, tatu: Better create context if one was not passed;
         *   mostly matters for root serialization types
         */
        if (context == null) {
            context = new TypeBindings(cls);
        }
        return _fromClass(cls, context);
    }
    // But if not, need to start resolving.
    if (type instanceof ParameterizedType) {
        return _fromParamType((ParameterizedType) type, context);
    }
    if (type instanceof GenericArrayType) {
        return _fromArrayType((GenericArrayType) type, context);
    }
    if (type instanceof TypeVariable<?>) {
        return _fromVariable((TypeVariable<?>) type, context);
    }
    if (type instanceof WildcardType) {
        return _fromWildcard((WildcardType) type, context);
    }
    // sanity check
    throw new IllegalArgumentException("Unrecognized Type: "+type.toString());
}

/**
 * This method deals with parameterized types, that is,
 * first class generic classes.
 *<p>
 * Since version 1.2, this resolves all parameterized types, not just
```

```
     * Maps or Collections.
     */
    protected JavaType _fromParamType(ParameterizedType type, TypeBindings context)
    {
        /* First: what is the actual base type? One odd thing
         * is that 'getRawType' returns Type, not Class<?> as
         * one might expect. But let's assume it is always of
         * type Class: if not, need to add more code to resolve
         * it to Class.
         */
        Class<?> rawType = (Class<?>) type.getRawType();
        Type[] args = type.getActualTypeArguments();
        int paramCount = (args == null) ? 0 : args.length;

        JavaType[] pt;

        if (paramCount == 0) {
            pt = NO_TYPES;
        } else {
            pt = new JavaType[paramCount];
            for (int i = 0; i < paramCount; ++i) {
                pt[i] = _fromType(args[i], context);
            }
        }

        // Ok: Map or Collection?
        if (Map.class.isAssignableFrom(rawType)) {
            JavaType subtype = _constructSimple(rawType, pt);
            JavaType[] mapParams = findParameterTypes(subtype, Map.class);
            if (mapParams.length != 2) {
                throw new IllegalArgumentException("Could not find 2 type parameters for Map class
"+rawType.getName()+" (found "+mapParams.length+")");
            }
            return MapType.construct(rawType, mapParams[0], mapParams[1]);
        }
        if (Collection.class.isAssignableFrom(rawType)) {
            JavaType subtype = _constructSimple(rawType, pt);
            JavaType[] collectionParams = findParameterTypes(subtype, Collection.class);
            if (collectionParams.length != 1) {
                throw new IllegalArgumentException("Could not find 1 type parameter for Collection class
"+rawType.getName()+" (found "+collectionParams.length+")");
            }
            return CollectionType.construct(rawType, collectionParams[0]);
        }
        if (paramCount == 0) { // no generics
            return new SimpleType(rawType);
        }
        return _constructSimple(rawType, pt);
```

```
}

protected static SimpleType _constructSimple(Class<?> rawType, JavaType[] parameterTypes)
{
   // Quick sanity check: must match numbers of types with expected...
   TypeVariable<?>[] typeVars = rawType.getTypeParameters();
   if (typeVars.length != parameterTypes.length) {
      throw new IllegalArgumentException("Parameter type mismatch for "+rawType.getName()
            +": expected "+typeVars.length+" parameters, was given "+parameterTypes.length);
   }
   String[] names = new String[typeVars.length];
   for (int i = 0, len = typeVars.length; i < len; ++i) {
      names[i] = typeVars[i].getName();
   }
   return new SimpleType(rawType, names, parameterTypes);
}

protected JavaType _fromArrayType(GenericArrayType type, TypeBindings context)
{
   JavaType compType = _fromType(type.getGenericComponentType(), context);
   return ArrayType.construct(compType);
}

protected JavaType _fromVariable(TypeVariable<?> type, TypeBindings context)
{
   /* 26-Sep-2009, tatus: It should be possible to try "partial"
    *  resolution; meaning that it is ok not to find bindings.
    *  For now this is indicated by passing null context.
    */
   if (context == null) {
      return _unknownType();
   }

   // Ok: here's where context might come in handy!
   String name = type.getName();
   JavaType actualType = context.findType(name);
   if (actualType != null) {
      return actualType;
   }

   /* 29-Jan-2010, tatu: We used to throw exception here, if type was
    *   bound: but the problem is that this can occur for generic "base"
    *   method, overridden by sub-class. If so, we will want to ignore
    *   current type (for method) since it will be masked.
    */
   Type[] bounds = type.getBounds();

   // With type variables we must use bound information.
```

```
        // Theoretically this gets tricky, as there may be multiple
        // bounds ("... extends A & B"); and optimally we might
        // want to choose the best match. Also, bounds are optional;
        // but here we are lucky in that implicit "Object" is
        // added as bounds if so.
        // Either way let's just use the first bound, for now, and
        // worry about better match later on if there is need.

        /* 29-Jan-2010, tatu: One more problem are recursive types
         *   (T extends Comparable<T>). Need to add "placeholder"
         *   for resolution to catch those.
         */
        context._addPlaceholder(name);
        return _fromType(bounds[0], context);
    }

    protected JavaType _fromWildcard(WildcardType type, TypeBindings context)
    {
        /* Similar to challenges with TypeVariable, we may have
         * multiple upper bounds. But it is also possible that if
         * upper bound defaults to Object, we might want to consider
         * lower bounds instead.
         *
         * For now, we won't try anything more advanced; above is
         * just for future reference.
         */
        return _fromType(type.getUpperBounds()[0], context);
    }

    private JavaType _mapType(Class<?> rawClass)
    {
        JavaType[] typeParams = findParameterTypes(rawClass, Map.class);
        // ok to have no types ("raw")
        if (typeParams == null) {
            return MapType.construct(rawClass, _unknownType(), _unknownType());
        }
        // but exactly 2 types if any found
        if (typeParams.length != 2) {
            throw new IllegalArgumentException("Strange Map type "+rawClass.getName()+": can not determine type
parameters");
        }
        return MapType.construct(rawClass, typeParams[0], typeParams[1]);
    }

    private JavaType _collectionType(Class<?> rawClass)
    {
        JavaType[] typeParams = findParameterTypes(rawClass, Collection.class);
        // ok to have no types ("raw")
```

```
        if (typeParams == null) {
            return CollectionType.construct(rawClass, _unknownType());
        }
        // but exactly 2 types if any found
        if (typeParams.length != 1) {
            throw new IllegalArgumentException("Strange Collection type "+rawClass.getName()+": can not determine type parameters");
        }
        return CollectionType.construct(rawClass, typeParams[0]);
    }

    protected static JavaType _resolveVariableViaSubTypes(HierarchicType leafType, String variableName, TypeBindings bindings)
    {
        // can't resolve raw types; possible to have as-of-yet-unbound types too:
        if (leafType != null && leafType.isGeneric()) {
            TypeVariable<?>[] typeVariables = leafType.getRawClass().getTypeParameters();
            for (int i = 0, len = typeVariables.length; i < len; ++i) {
                TypeVariable<?> tv = typeVariables[i];
                if (variableName.equals(tv.getName())) {
                    // further resolution needed?
                    Type type = leafType.asGeneric().getActualTypeArguments()[i];
                    if (type instanceof TypeVariable<?>) {
                        return _resolveVariableViaSubTypes(leafType.getSubType(), ((TypeVariable<?>) type).getName(), bindings);
                    }
                    // no we're good for the variable (but it may have parameterization of its own)
                    return instance._fromType(type, bindings);
                }
            }
        }
        return instance._unknownType();
    }

    protected JavaType _unknownType() {
        return _fromClass(Object.class, null);
    }

    /**
     * Helper method used to find inheritance (implements, extends) path
     * between given types, if one exists (caller generally checks before
     * calling this method). Returned type represents given <b>subtype</b>,
     * with supertype linkage extending to <b>supertype</b>.
     */
    protected static HierarchicType  _findSuperTypeChain(Class<?> subtype, Class<?> supertype)
    {
        // If super-type is a class (not interface), bit simpler
        if (supertype.isInterface()) {
```

```
        return _findSuperInterfaceChain(subtype, supertype);
    }
    return _findSuperClassChain(subtype, supertype);
}

protected static HierarchicType _findSuperClassChain(Type currentType, Class<?> target)
{
    HierarchicType current = new HierarchicType(currentType);
    Class<?> raw = current.getRawClass();
    if (raw == target) {
        return current;
    }
    // Otherwise, keep on going down the rat hole...
    Type parent = raw.getGenericSuperclass();
    if (parent != null) {
        HierarchicType sup = _findSuperClassChain(parent, target);
        if (sup != null) {
            sup.setSubType(current);
            current.setSuperType(sup);
            return current;
        }
    }
    return null;
}

protected static HierarchicType _findSuperInterfaceChain(Type currentType, Class<?> target)
{
    HierarchicType current = new HierarchicType(currentType);
    Class<?> raw = current.getRawClass();
    if (raw == target) {
        return current;
    }
    // Otherwise, keep on going down the rat hole; first implemented interfaces
    Type[] parents = raw.getGenericInterfaces();
    // as long as there are superclasses
    // and unless we have already seen the type (<T extends X<T>>)
    if (parents != null) {
        for (Type parent : parents) {
            HierarchicType sup = _findSuperInterfaceChain(parent, target);
            if (sup != null) {
                sup.setSubType(current);
                current.setSuperType(sup);
                return current;
            }
        }
    }
    // and then super-class if any
    Type parent = raw.getGenericSuperclass();
```

```
        if (parent != null) {
            HierarchicType sup = _findSuperInterfaceChain(parent, target);
            if (sup != null) {
                sup.setSubType(current);
                current.setSuperType(sup);
                return current;
            }
        }
        return null;
    }

    /*
    protected static final Class<?> _typeToClass(Type type)
    {
        if (type instanceof Class<?>) {
            return (Class<?>) type;
        }
        if (type instanceof ParameterizedType) {
            return (Class<?>)((ParameterizedType) type).getRawType();
        }
        // we don't really support other types; GenericArrayType may or may not need support in future?
        throw new IllegalArgumentException("Can not coerce Type "+type.getClass().getName()+" into Class<?>");
    }
    */
}
package org.codehaus.jackson.map.type;

import org.codehaus.jackson.type.JavaType;

/**
* Type that represents Java Collection types (Lists, Sets).
*/
public final class CollectionType
    extends TypeBase
{
    /**
     * Type of elements in collection
     */
    final JavaType _elementType;

    /*
    /**********************************************************
    /* Life-cycle
    /**********************************************************
     */

    private CollectionType(Class<?> collT, JavaType elemT)
    {
```

```
    super(collT,  elemT.hashCode());
    _elementType = elemT;
}

@Override
protected JavaType _narrow(Class<?> subclass) {
    return new CollectionType(subclass, _elementType);
}

@Override
public JavaType narrowContentsBy(Class<?> contentClass)
{
    // Can do a quick check first:
    if (contentClass == _elementType.getRawClass()) {
        return this;
    }
    JavaType newElementType = _elementType.narrowBy(contentClass);
    return new CollectionType(_class, newElementType).copyHandlers(this);
}

public static CollectionType construct(Class<?> rawType, JavaType elemT)
{
    // nominally component types will be just Object.class
    return new CollectionType(rawType, elemT);
}

// Since 1.7:
@Override
public CollectionType withTypeHandler(Object h)
{
    CollectionType newInstance = new CollectionType(_class, _elementType);
    newInstance._typeHandler = h;
    return newInstance;
}

// Since 1.7:
@Override
public CollectionType withContentTypeHandler(Object h)
{
    return new CollectionType(_class, _elementType.withTypeHandler(h));
}

@Override
protected String buildCanonicalName() {
    StringBuilder sb = new StringBuilder();
    sb.append(_class.getName());
    if (_elementType != null) {
        sb.append('<');
```

```
        sb.append(_elementType.toCanonical());
        sb.append('>');
    }
    return sb.toString();
}

/*
/**********************************************************
/* Public API
/**********************************************************
 */

@Override
public JavaType getContentType() { return _elementType; }
@Override
public int containedTypeCount() { return 1; }
@Override
public JavaType containedType(int index) {
        return (index == 0) ? _elementType : null;
}

/**
 * Not sure if we should count on this, but type names
 * for core interfaces use "E" for element type
 */
@Override
public String containedTypeName(int index) {
    if (index == 0) return "E";
    return null;
}

@Override
public StringBuilder getErasedSignature(StringBuilder sb) {
    return _classSignature(_class, sb, true);
}

@Override
public StringBuilder getGenericSignature(StringBuilder sb) {
    _classSignature(_class, sb, false);
    sb.append('<');
    _elementType.getGenericSignature(sb);
    sb.append(">;");
    return sb;
}

/*
/**********************************************************
/* Extended API
```

```
  /**********************************************************
   */

  @Override
  public boolean isContainerType() { return true; }

  /*
  /**********************************************************
  /* Standard methods
  /**********************************************************
   */

  @Override
    public String toString()
  {
    return "[collection type; class "+_class.getName()+", contains "+_elementType+"]";
  }

  @Override
  public boolean equals(Object o)
  {
    if (o == this) return true;
    if (o == null) return false;
    if (o.getClass() != getClass()) return false;

    CollectionType other = (CollectionType) o;
    return  (_class == other._class)
       && _elementType.equals(other._elementType);
  }
}
package org.codehaus.jackson.map.type;

import java.util.*;

import org.codehaus.jackson.type.JavaType;

/**
* Simple recursive-descent parser for parsing canonical {@link JavaType}
* representations and constructing type instances.
*
* @author tatu
* @since 1.5
*/
public class TypeParser
{
  final TypeFactory _factory;

  public TypeParser(TypeFactory f) {
```

```
    _factory = f;
}

public JavaType parse(String canonical)
    throws IllegalArgumentException
{
    canonical = canonical.trim();
    MyTokenizer tokens = new MyTokenizer(canonical);
    JavaType type = parseType(tokens);
    // must be end, now
    if (tokens.hasMoreTokens()) {
        throw _problem(tokens, "Unexpected tokens after complete type");
    }
    return type;
}

protected JavaType parseType(MyTokenizer tokens)
    throws IllegalArgumentException
{
    if (!tokens.hasMoreTokens()) {
        throw _problem(tokens, "Unexpected end-of-string");
    }
    Class<?> base = findClass(tokens.nextToken(), tokens);
    // either end (ok, non generic type), or generics
    if (tokens.hasMoreTokens()) {
        String token = tokens.nextToken();
        if ("<".equals(token)) {
            return _factory._fromParameterizedClass(base, parseTypes(tokens));
        }
        // can be comma that separates types, or closing '>'
        tokens.pushBack(token);
    }
    return _factory._fromClass(base, null);
}

protected List<JavaType> parseTypes(MyTokenizer tokens)
    throws IllegalArgumentException
{
    ArrayList<JavaType> types = new ArrayList<JavaType>();
    while (tokens.hasMoreTokens()) {
        types.add(parseType(tokens));
        if (!tokens.hasMoreTokens()) break;
        String token = tokens.nextToken();
        if (">".equals(token)) return types;
        if (!",".equals(token)) {
            throw _problem(tokens, "Unexpected token '"+token+"', expected ',' or '>')");
        }
    }
```

```
        throw _problem(tokens, "Unexpected end-of-string");
    }

    protected Class<?> findClass(String className, MyTokenizer tokens)
    {
        try {
            /* [JACKSON-350]: Default Class.forName() won't work too well; context class loader
             *    seems like slightly better choice
             */
//          return Class.forName(className);
            ClassLoader loader = Thread.currentThread().getContextClassLoader();
            return Class.forName(className, true, loader);
        } catch (Exception e) {
            if (e instanceof RuntimeException) {
                throw (RuntimeException) e;
            }
            throw _problem(tokens, "Can not locate class '"+className+"', problem: "+e.getMessage());
        }
    }

    protected IllegalArgumentException _problem(MyTokenizer tokens, String msg)
    {
        return new IllegalArgumentException("Failed to parse type '"+tokens.getAllInput()
                +"' (remaining: '"+tokens.getRemainingInput()+"'): "+msg);
    }

    final static class MyTokenizer
        extends StringTokenizer
    {
        protected final String _input;

        protected int _index;

        protected String _pushbackToken;

        public MyTokenizer(String str) {
            super(str, "<,>", true);
            _input = str;
        }

        @Override
        public boolean hasMoreTokens() {
            return (_pushbackToken != null) || super.hasMoreTokens();
        }

        @Override
        public String nextToken() {
            String token;
```

```
            if (_pushbackToken != null) {
                token = _pushbackToken;
                _pushbackToken = null;
            } else {
                token = super.nextToken();
            }
            _index += token.length();
            return token;
        }

        public void pushBack(String token) {
            _pushbackToken = token;
            _index -= token.length();
        }

        public String getAllInput() { return _input; }
        public String getUsedInput() { return _input.substring(0, _index); }
        public String getRemainingInput() { return _input.substring(_index); }
    }
}
package org.codehaus.jackson.map.type;

/**
* Key class, used as an efficient and accurate key
* for locating per-class values, such as
* {@link org.codehaus.jackson.map.JsonSerializer}s.
*<p>
* The reason for having a separate key class instead of
* directly using {@link Class} as key is mostly
* to allow for redefining <code>hashCode</code> method --
* for some strange reason, {@link Class} does not
* redefine {@link Object#hashCode} and thus uses identity
* hash, which is pretty slow. This makes key access using
* {@link Class} unnecessarily slow.
*<p>
* Note: since class is not strictly immutable, caller must
* know what it is doing, if changing field values.
*/
public final class ClassKey
    implements Comparable<ClassKey>
{
    String _className;

    Class<?> _class;

    /**
     * Let's cache hash code straight away, since we are
     * almost certain to need it.
```

```
 */
int _hashCode;

public ClassKey()
{
   _class = null;
   _className = null;
   _hashCode = 0;
}

public ClassKey(Class<?> clz)
{
   _class = clz;
   _className = clz.getName();
   _hashCode = _className.hashCode();
}

public void reset(Class<?> clz)
{
   _class = clz;
   _className = clz.getName();
   _hashCode = _className.hashCode();
}

/*
/************************************************************
/* Comparable
/************************************************************
 */

public int compareTo(ClassKey other)
{
   // Just need to sort by name, ok to collide (unless used in TreeMap/Set!)
   return _className.compareTo(other._className);
}

/*
/************************************************************
/* Standard methods
/************************************************************
 */

@Override
   public boolean equals(Object o)
{
   if (o == this) return true;
   if (o == null) return false;
   if (o.getClass() != getClass()) return false;
```

```
        ClassKey other = (ClassKey) o;

        /* Is it possible to have different Class object for same name + class loader combo?
         * Let's assume answer is no: if this is wrong, will need to uncomment following functionality
         */
        /*
        return (other._className.equals(_className))
            && (other._class.getClassLoader() == _class.getClassLoader());
        */
        return other._class == _class;
    }

    @Override public int hashCode() { return _hashCode; }

    @Override public String toString() { return _className; }

}
package org.codehaus.jackson.map.type;

import org.codehaus.jackson.type.JavaType;

/**
* Type that represents Java Map types.
*/
public final class MapType
    extends TypeBase
{
    /**
     * Type of keys of Map.
     */
    final JavaType _keyType;

    /**
     * Type of values of Map.
     */
    final JavaType _valueType;

    /*
    /**********************************************************
    /* Life-cycle
    /**********************************************************
     */

    private MapType(Class<?> mapType, JavaType keyT, JavaType valueT)
    {
        super(mapType, keyT.hashCode() ^ valueT.hashCode());
        _keyType = keyT;
        _valueType = valueT;
```

```
}

public static MapType construct(Class<?> rawType, JavaType keyT, JavaType valueT)
{
   // nominally component types will be just Object.class
   return new MapType(rawType, keyT, valueT);
}

@Override
protected JavaType _narrow(Class<?> subclass)
{
   return new MapType(subclass, _keyType, _valueType);
}

@Override
public JavaType narrowContentsBy(Class<?> contentClass)
{
   // Can do a quick check first:
   if (contentClass == _valueType.getRawClass()) {
      return this;
   }
   JavaType newValueType = _valueType.narrowBy(contentClass);
   return new MapType(_class, _keyType, newValueType).copyHandlers(this);
}

public JavaType narrowKey(Class<?> keySubclass)
{
   // Can do a quick check first:
   if (keySubclass == _keyType.getRawClass()) {
      return this;
   }
   JavaType newKeyType = _keyType.narrowBy(keySubclass);
   return new MapType(_class, newKeyType, _valueType).copyHandlers(this);
}

// Since 1.7:
@Override
public MapType withTypeHandler(Object h)
{
   MapType newInstance = new MapType(_class, _keyType, _valueType);
   newInstance._typeHandler = h;
   return newInstance;
}

// Since 1.7:
@Override
public MapType withContentTypeHandler(Object h)
{
```

```
    return new MapType(_class, _keyType, _valueType.withTypeHandler(h));
}

@Override
protected String buildCanonicalName() {
    StringBuilder sb = new StringBuilder();
    sb.append(_class.getName());
    if (_keyType != null) {
        sb.append('<');
        sb.append(_keyType.toCanonical());
        sb.append(',');
        sb.append(_valueType.toCanonical());
        sb.append('>');
    }
    return sb.toString();
}

/*
/**********************************************************
/* Public API
/**********************************************************
 */

@Override
public boolean isContainerType() { return true; }

@Override
public JavaType getKeyType() { return _keyType; }

@Override
public JavaType getContentType() { return _valueType; }

@Override
public int containedTypeCount() { return 2; }

@Override
public JavaType containedType(int index) {
    if (index == 0) return _keyType;
    if (index == 1) return _valueType;
    return null;
}

/**
 * Not sure if we should count on this, but type names
 * for core interfaces are "K" and "V" respectively.
 * For now let's assume this should work.
 */
@Override
```

```
    public String containedTypeName(int index) {
        if (index == 0) return "K";
        if (index == 1) return "V";
        return null;
    }

    @Override
    public StringBuilder getErasedSignature(StringBuilder sb) {
        return _classSignature(_class, sb, true);
    }

    @Override
    public StringBuilder getGenericSignature(StringBuilder sb)
    {
        _classSignature(_class, sb, false);
        sb.append('<');
        _keyType.getGenericSignature(sb);
        _valueType.getGenericSignature(sb);
        sb.append(">;");
        return sb;
    }

    /*
    /**********************************************************
    /* Standard methods
    /**********************************************************
     */

    @Override
    public String toString()
    {
        return "[map type; class "+_class.getName()+", "+_keyType+" -> "+_valueType+"]";
    }

    @Override
    public boolean equals(Object o)
    {
        if (o == this) return true;
        if (o == null) return false;
        if (o.getClass() != getClass()) return false;

        MapType other = (MapType) o;
        return (_class == other._class)
            && _keyType.equals(other._keyType)
            && _valueType.equals(other._valueType);
    }
}
```

```
package org.codehaus.jackson.map.type;

import java.lang.reflect.*;
import java.util.*;

import org.codehaus.jackson.type.JavaType;

/**
* Helper class used for resolving type parameters for given class
*
* @since 1.5
*/
public class TypeBindings
{
  private final static JavaType[] NO_TYPES = new JavaType[0];

  /**
   * Marker to use for (temporarily) unbound references.
   */
  public final static JavaType UNBOUND = new SimpleType(Object.class);

  /**
   * Context type used for resolving all types, if specified. May be null,
   * in which case {@link #_contextClass} is used instead.
   */
  protected final JavaType _contextType;

  /**
   * Specific class to use for resolving all types, for methods and fields
   * class and its superclasses and -interfaces contain.
   */
  protected final Class<?> _contextClass;

  /**
   * Lazily-instantiated bindings of resolved type parameters
   */
  protected Map<String,JavaType> _bindings;

  /**
   * Also: we may temporarily want to mark certain named types
   * as resolved (but without exact type); if so, we'll just store
   * names here.
   */
  protected HashSet<String> _placeholders;

  /**
   * Sometimes it is necessary to allow hierarchic resolution of types: specifically
   * in cases where there are local bindings (for methods, constructors). If so,
```

```
     * we'll just use simple delegation model.
     *
     * @since 1.7
     */
    private final TypeBindings _parentBindings;

    /*
    /**********************************************************
    /* Construction
    /**********************************************************
     */

    public TypeBindings(Class<?> cc)
    {
        this(null, cc, null);
    }

    public TypeBindings(JavaType type)
    {
        this(null, type.getRawClass(), type);
    }

    /**
     * Constructor used to create "child" instances; mostly to
     * allow delegation from explicitly defined local overrides
     * (local type variables for methods, constructors) to
     * contextual (class-defined) ones.
     *
     * @since 1.7
     */
    public TypeBindings childInstance() {
        return new TypeBindings(this, _contextClass, _contextType);
    }

    /**
     * @since 1.7
     */
    private TypeBindings(TypeBindings parent, Class<?> cc, JavaType type)
    {
        _parentBindings = parent;
        _contextClass = cc;
        _contextType = type;
    }

    /*
    /**********************************************************
    /* Accesors
    /**********************************************************
```

```
 */

public int getBindingCount() {
    if (_bindings == null) {
        _resolve();
    }
    return _bindings.size();
}

public JavaType findType(String name)
{
    if (_bindings == null) {
        _resolve();
    }
    JavaType t = _bindings.get(name);
    if (t != null) {
        return t;
    }
    if (_placeholders != null && _placeholders.contains(name)) {
        return UNBOUND;
    }
    // New with 1.7: check parent context
    if (_parentBindings != null) {
        return _parentBindings.findType(name);
    }
    // nothing found, so...
    // Should we throw an exception or just return null?

    /* [JACKSON-499] 18-Feb-2011, tatu: There are some tricky type bindings within
     *   java.util, such as HashMap$KeySet; so let's punt the problem
     *   (honestly not sure what to do -- they are unbound for good, I think)
     */
    if (_contextClass != null) {
        Class<?> enclosing = _contextClass.getEnclosingClass();
        if (enclosing != null) {
            Package pkg = enclosing.getPackage();
            if (pkg != null) {
                if (pkg.getName().equals("java.util")) {
                    return UNBOUND;
                }
            }
        }
    }

    String className;
    if (_contextClass != null) {
        className = _contextClass.getName();
    } else if (_contextType != null) {
```

```
            className = _contextType.toString();
        } else {
            className = "UNKNOWN";
        }
        throw new IllegalArgumentException("Type variable '"+name
                +"' can not be resolved (with context of class "+className+")");
        //t = UNBOUND;
    }

    public void addBinding(String name, JavaType type)
    {
        if (_bindings == null) {
            _bindings = new LinkedHashMap<String,JavaType>();
        }
        _bindings.put(name, type);
    }

    public JavaType[] typesAsArray()
    {
        if (_bindings == null) {
            _resolve();
        }
        if (_bindings.size() == 0) {
            return NO_TYPES;
        }
        return _bindings.values().toArray(new JavaType[_bindings.size()]);
    }

    /*
    /**********************************************************
    /* Internal methods
    /**********************************************************
     */

    protected void _resolve()
    {
        _resolveBindings(_contextClass);

        // finally: may have root level type info too
        if (_contextType != null) {
            int count = _contextType.containedTypeCount();
            if (count > 0) {
                if (_bindings == null) {
                    _bindings = new LinkedHashMap<String,JavaType>();
                }
                for (int i = 0; i < count; ++i) {
                    String name = _contextType.containedTypeName(i);
                    JavaType type = _contextType.containedType(i);
```

```
                _bindings.put(name, type);
            }
        }
    }

    // nothing bound? mark with empty map to prevent further calls
    if (_bindings == null) {
        _bindings = Collections.emptyMap();
    }
}

public void _addPlaceholder(String name) {
    if (_placeholders == null) {
        _placeholders = new HashSet<String>();
    }
    _placeholders.add(name);
}

protected void _resolveBindings(Type t)
{
    if (t == null) return;

    Class<?> raw;
    if (t instanceof ParameterizedType) {
        ParameterizedType pt = (ParameterizedType) t;
        Type[] args = pt.getActualTypeArguments();
        if (args  != null && args.length > 0) {
            Class<?> rawType = (Class<?>) pt.getRawType();
            TypeVariable<?>[] vars = rawType.getTypeParameters();
            if (vars.length != args.length) {
                throw new IllegalArgumentException("Strange parametrized type (in class "+rawType.getName()+"):
number of type arguments != number of type parameters ("+args.length+" vs "+vars.length+")");
            }
            for (int i = 0, len = args.length; i < len; ++i) {
                TypeVariable<?> var = vars[i];
                String name = var.getName();
                if (_bindings == null) {
                    _bindings = new LinkedHashMap<String,JavaType>();
                } else {
                    /* 24-Mar-2010, tatu: Better ensure that we do not overwrite something
                     *  collected earlier (since we descend towards super-classes):
                     */
                    if (_bindings.containsKey(name)) continue;
                }
                // first: add a placeholder to prevent infinite loops
                _addPlaceholder(name);
                // then resolve type
                _bindings.put(name, TypeFactory.instance._fromType(args[i], this));
```

```
            }
        }
        raw = (Class<?>)pt.getRawType();
    } else if (t instanceof Class<?>) {
        raw = (Class<?>) t;
        /* 24-Mar-2010, tatu: Can not have true generics definitions, but can
         *   have lower bounds ("<T extends BeanBase>") in declaration itself
         */
        TypeVariable<?>[] vars = raw.getTypeParameters();
        if (vars != null && vars.length > 0) {
            for (TypeVariable<?> var : vars) {
                String name = var.getName();
                Type varType = var.getBounds()[0];
                if (varType != null) {
                    if (_bindings == null) {
                        _bindings = new LinkedHashMap<String,JavaType>();
                    } else { // and no overwriting...
                        if (_bindings.containsKey(name)) continue;
                    }
                    _addPlaceholder(name); // to prevent infinite loops
                    _bindings.put(name, TypeFactory.instance._fromType(varType, this));
                }
            }
        }
    } else { // probably can't be any of these... so let's skip for now
        //if (type instanceof GenericArrayType) {
        //if (type instanceof TypeVariable<?>) {
        // if (type instanceof WildcardType) {
        return;
    }
    // but even if it's not a parameterized type, its super types may be:
    _resolveBindings(raw.getGenericSuperclass());
    for (Type intType : raw.getGenericInterfaces()) {
        _resolveBindings(intType);
    }
}

@Override
public String toString()
{
    if (_bindings == null) {
        _resolve();
    }
    StringBuilder sb = new StringBuilder("[TypeBindings for ");
    if (_contextType != null) {
        sb.append(_contextType.toString());
    } else {
        sb.append(_contextClass.getName());
```

```
        }
        sb.append(": ").append(_bindings).append("]");
        return sb.toString();
    }
}
package org.codehaus.jackson.map.type;

import org.codehaus.jackson.type.JavaType;

public abstract class TypeBase extends JavaType
{
    /**
     * Lazily initialized external representation of the type
     */
    volatile String _canonicalName;

    protected TypeBase(Class<?> raw, int hash) {
        super(raw, hash);
    }

    @Override
    public String toCanonical()
    {
     String str = _canonicalName;
     if (str == null) {
            str = buildCanonicalName();
     }
     return str;
    }

    protected abstract String buildCanonicalName();

    protected final JavaType copyHandlers(JavaType fromType)
    {
        _valueHandler = fromType.getValueHandler();
        _typeHandler = fromType.getTypeHandler();
        return this;
    }

    @Override
    public abstract StringBuilder getGenericSignature(StringBuilder sb);

    @Override
    public abstract StringBuilder getErasedSignature(StringBuilder sb);

    /*
    /**********************************************************
    /* Methods for sub-classes to use
```

```
/**********************************************************
 */

/**
 * @param trailingSemicolon Whether to add trailing semicolon for non-primitive
 *   (reference) types or not
 */
protected static StringBuilder _classSignature(Class<?> cls, StringBuilder sb,
      boolean trailingSemicolon)
{
   if (cls.isPrimitive()) {
      if (cls == Boolean.TYPE) {
         sb.append('Z');
      } else if (cls == Byte.TYPE) {
         sb.append('B');
      }
      else if (cls == Short.TYPE) {
         sb.append('S');
      }
      else if (cls == Character.TYPE) {
         sb.append('C');
      }
      else if (cls == Integer.TYPE) {
         sb.append('I');
      }
      else if (cls == Long.TYPE) {
         sb.append('J');
      }
      else if (cls == Float.TYPE) {
         sb.append('F');
      }
      else if (cls == Double.TYPE) {
         sb.append('D');
      }
      else if (cls == Void.TYPE) {
         sb.append('V');
      } else {
         throw new IllegalStateException("Unrecognized primitive type: "+cls.getName());
      }
   } else {
      sb.append('L');
      String name = cls.getName();
      for (int i = 0, len = name.length(); i < len; ++i) {
         char c = name.charAt(i);
         if (c == '.') c = '/';
         sb.append(c);
      }
      if (trailingSemicolon) {
```

```
            sb.append(';');
          }
        }
        return sb;
    }
}
package org.codehaus.jackson.map.type;

/*
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
import java.lang.reflect.TypeVariable;
*/

import java.util.*;

import org.codehaus.jackson.type.JavaType;

/**
* Simple types are defined as anything other than one of recognized
* container types (arrays, Collections, Maps). For our needs we
* need not know anything further, since we have no way of dealing
* with generic types other than Collections and Maps.
*/
public final class SimpleType
    extends TypeBase
{
    /**
     * Generic type arguments for this type.
     */
    protected final JavaType[] _typeParameters;

    /**
     * Names of generic type arguments for this type; will
     * match values in {@link #_typeParameters}
     */
    protected final String[] _typeNames;

    /*
    /**********************************************************
    /* Life-cycle
    /**********************************************************
     */

    protected SimpleType(Class<?> cls) {
        this(cls, null, null);
    }
```

```
  protected SimpleType(Class<?> cls,
                       String[] typeNames, JavaType[] typeParams) {
      super(cls, 0);
      if (typeNames == null || typeNames.length == 0) {
          _typeNames = null;
          _typeParameters = null;
      } else {
          _typeNames = typeNames;
          _typeParameters = typeParams;
      }
  }

  @Override
  protected JavaType _narrow(Class<?> subclass)
  {
      // Should we check that there is a sub-class relationship?
      return new SimpleType(subclass, _typeNames, _typeParameters);
  }

  @Override
  public JavaType narrowContentsBy(Class<?> subclass)
  {
      // should never get called
      throw new IllegalArgumentException("Internal error: SimpleType.narrowContentsBy() should never be
called");
  }

  public static SimpleType construct(Class<?> cls)
  {
      /* Let's add sanity checks, just to ensure no
       * Map/Collection entries are constructed
       */
      if (Map.class.isAssignableFrom(cls)) {
          throw new IllegalArgumentException("Can not construct SimpleType for a Map (class:
"+cls.getName()+")");
      }
      if (Collection.class.isAssignableFrom(cls)) {
          throw new IllegalArgumentException("Can not construct SimpleType for a Collection (class:
"+cls.getName()+")");
      }
      // ... and while we are at it, not array types either
      if (cls.isArray()) {
          throw new IllegalArgumentException("Can not construct SimpleType for an array (class:
"+cls.getName()+")");
      }
      return new SimpleType(cls);
  }
```

```
  // Since 1.7:
  @Override
  public SimpleType withTypeHandler(Object h)
  {
     SimpleType newInstance = new SimpleType(_class, _typeNames, _typeParameters);
     newInstance._typeHandler = h;
     return newInstance;
  }

  // Since 1.7:
  @Override
  public JavaType withContentTypeHandler(Object h)
  {
     // no content type, so:
     throw new IllegalArgumentException("Simple types have no content types; can not call
withContenTypeHandler()");
  }

  @Override
  protected String buildCanonicalName()
  {
     StringBuilder sb = new StringBuilder();
     sb.append(_class.getName());
     if (_typeParameters != null && _typeParameters.length > 0) {
        sb.append('<');
        boolean first = true;
        for (JavaType t : _typeParameters) {
           if (first) {
              first = false;
           } else {
              sb.append(',');
           }
           sb.append(t.toCanonical());
        }
        sb.append('>');
     }
     return sb.toString();
  }

  /*
  /**********************************************************
  /* Public API
  /**********************************************************
   */

  @Override
  public boolean isContainerType() { return false; }
```

```
@Override
public int containedTypeCount() {
   return (_typeParameters == null) ? 0 : _typeParameters.length;
}

@Override
public JavaType containedType(int index)
{
   if (index < 0 || _typeParameters == null || index >= _typeParameters.length) {
      return null;
   }
   return _typeParameters[index];
}

@Override
public String containedTypeName(int index)
{
   if (index < 0 || _typeNames == null || index >= _typeNames.length) {
      return null;
   }
   return _typeNames[index];
}

@Override
public StringBuilder getErasedSignature(StringBuilder sb) {
   return _classSignature(_class, sb, true);
}

@Override
public StringBuilder getGenericSignature(StringBuilder sb)
{
   _classSignature(_class, sb, false);
   if (_typeParameters != null) {
      sb.append('<');
      for (JavaType param : _typeParameters) {
         sb = param.getGenericSignature(sb);
      }
      sb.append('>');
   }
   sb.append(';');
   return sb;
}

/*
/**********************************************************
/* Standard methods
/**********************************************************
 */
```

```
  @Override
  public String toString()
  {
     StringBuilder sb = new StringBuilder(40);
     sb.append("[simple type, class ").append(buildCanonicalName()).append(']');
     return sb.toString();
  }

  @Override
  public boolean equals(Object o)
  {
     if (o == this) return true;
     if (o == null) return false;
     if (o.getClass() != getClass()) return false;

     SimpleType other = (SimpleType) o;

     // Classes must be identical...
     if (other._class != this._class) return false;

     // And finally, generic bindings, if any
     JavaType[] p1 = _typeParameters;
     JavaType[] p2 = other._typeParameters;
     if (p1 == null) {
        return (p2 == null) || p2.length == 0;
     }
     if (p2 == null) return false;

     if (p1.length != p2.length) return false;
     for (int i = 0, len = p1.length; i < len; ++i) {
        if (!p1[i].equals(p2[i])) {
           return false;
        }
     }
     return true;
  }
}
package org.codehaus.jackson.map;

/**
* Interface used to indicate deserializers that want to do post-processing
* after construction and being added to {@link DeserializerProvider},
* but before being used. This is typically used to resolve references
* to other contained types; for example, bean deserializers use this
* to eagerly find deserializers for contained field types.
*/
public interface ResolvableDeserializer
```

```
{
  /**
   * Method called after {@link DeserializerProvider} has registered
   * the deserializer, but before it has returned it to the caller.
   * Called object can then resolve its dependencies to other types,
   * including self-references (direct or indirect).
   *
   * @param provider Provider that has constructed deserializer this method
   *   is called on.
   */
  public abstract void resolve(DeserializationConfig config, DeserializerProvider provider)
     throws JsonMappingException;
}
package org.codehaus.jackson.map.ext;

import java.io.IOException;
import java.util.*;

import org.joda.time.DateMidnight;
import org.joda.time.DateTime;
import org.joda.time.DateTimeZone;
import org.joda.time.LocalDate;
import org.joda.time.LocalDateTime;
import org.joda.time.ReadableDateTime;
import org.joda.time.ReadableInstant;
import org.joda.time.format.DateTimeFormatter;
import org.joda.time.format.ISODateTimeFormat;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.deser.StdDeserializer;
import org.codehaus.jackson.map.deser.StdScalarDeserializer;
import org.codehaus.jackson.map.util.Provider;

/**
* Provider for deserializers that handle some basic data types
* for <a href="http://joda-time.sourceforge.net/">Joda</a> date/time library.
*
* @since 1.4
*/
public class JodaDeserializers
  implements Provider<StdDeserializer<?>>
{
  public Collection<StdDeserializer<?>> provide() {
     return Arrays.asList(new StdDeserializer<?>[] {
           new DateTimeDeserializer<DateTime>(DateTime.class)
           ,new DateTimeDeserializer<ReadableDateTime>(ReadableDateTime.class)
           ,new DateTimeDeserializer<ReadableInstant>(ReadableInstant.class)
```

```
            ,new LocalDateDeserializer()
            ,new LocalDateTimeDeserializer()
            ,new DateMidnightDeserializer()
        });
    }

    /*
    /**********************************************************************
    /* Intermediate base classes
    /**********************************************************************
    */

    abstract static class JodaDeserializer<T> extends StdScalarDeserializer<T>
    {
        final static DateTimeFormatter _localDateTimeFormat = ISODateTimeFormat.localDateOptionalTimeParser();

        protected JodaDeserializer(Class<T> cls) { super(cls); }

        protected DateTime parseLocal(JsonParser jp)
            throws IOException, JsonProcessingException
        {
            String str = jp.getText().trim();
            if (str.length() == 0) { // [JACKSON-360]
                return null;
            }
            return _localDateTimeFormat.parseDateTime(str);
        }
    }

    /*
    /**********************************************************************
    /* Concrete deserializers
    /**********************************************************************
    */

    /**
     * Basic deserializer for {@link DateTime}. Accepts JSON String and Number
     * values and passes those to single-argument constructor.
     * Does not (yet?) support JSON object; support can be added if desired.
     *<p>
     * Since 1.6 this has been generic, to handle multiple related types,
     * including super types of {@link DateTime}
     */
    public static class DateTimeDeserializer<T extends ReadableInstant>
        extends JodaDeserializer<T>
    {
        public DateTimeDeserializer(Class<T> cls) { super(cls); }
```

```
    @SuppressWarnings("unchecked")
    @Override
    public T deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();
        if (t == JsonToken.VALUE_NUMBER_INT) {
            return (T) new DateTime(jp.getLongValue(), DateTimeZone.UTC);
        }
        if (t == JsonToken.VALUE_STRING) {
            String str = jp.getText().trim();
            if (str.length() == 0) { // [JACKSON-360]
                return null;
            }
            return (T) new DateTime(str, DateTimeZone.UTC);
        }
        throw ctxt.mappingException(getValueClass());
    }
}

/**
 * @since 1.5
 */
public static class LocalDateDeserializer
    extends JodaDeserializer<LocalDate>
{
    public LocalDateDeserializer() { super(LocalDate.class); }

    @Override
    public LocalDate deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        // We'll accept either long (timestamp) or array:
        if (jp.isExpectedStartArrayToken()) {
            jp.nextToken(); // VALUE_NUMBER_INT
            int year = jp.getIntValue();
            jp.nextToken(); // VALUE_NUMBER_INT
            int month = jp.getIntValue();
            jp.nextToken(); // VALUE_NUMBER_INT
            int day = jp.getIntValue();
            if (jp.nextToken() != JsonToken.END_ARRAY) {
                ctxt.wrongTokenException(jp, JsonToken.END_ARRAY, "after LocalDate ints");
            }
            return new LocalDate(year, month, day);
        }
        switch (jp.getCurrentToken()) {
        case VALUE_NUMBER_INT:
            return new LocalDate(jp.getLongValue());
```

```
        case VALUE_STRING:
            DateTime local = parseLocal(jp);
            if (local == null) {
                return null;
            }
            return local.toLocalDate();
        }
        ctxt.wrongTokenException(jp, JsonToken.START_ARRAY, "expected JSON Array, String or Number");
        return null;
    }
}

/**
 * @since 1.5
 */
public static class LocalDateTimeDeserializer
    extends JodaDeserializer<LocalDateTime>
{
    public LocalDateTimeDeserializer() { super(LocalDateTime.class); }

    @Override
    public LocalDateTime deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        // We'll accept either long (timestamp) or array:
        if (jp.isExpectedStartArrayToken()) {
            jp.nextToken(); // VALUE_NUMBER_INT
            int year = jp.getIntValue();
            jp.nextToken(); // VALUE_NUMBER_INT
            int month = jp.getIntValue();
            jp.nextToken(); // VALUE_NUMBER_INT
            int day = jp.getIntValue();
            jp.nextToken(); // VALUE_NUMBER_INT
            int hour = jp.getIntValue();
            jp.nextToken(); // VALUE_NUMBER_INT
            int minute = jp.getIntValue();
            jp.nextToken(); // VALUE_NUMBER_INT
            int second = jp.getIntValue();
            // let's leave milliseconds optional?
            int millisecond = 0;
            if (jp.nextToken() != JsonToken.END_ARRAY) { // VALUE_NUMBER_INT
                millisecond = jp.getIntValue();
                jp.nextToken(); // END_ARRAY?
            }
            if (jp.getCurrentToken() != JsonToken.END_ARRAY) {
                ctxt.wrongTokenException(jp, JsonToken.END_ARRAY, "after LocalDateTime ints");
            }
            return new LocalDateTime(year, month, day, hour, minute, second, millisecond);
```

```
        }

        switch (jp.getCurrentToken()) {
        case VALUE_NUMBER_INT:
            return new LocalDateTime(jp.getLongValue());
        case VALUE_STRING:
            DateTime local = parseLocal(jp);
            if (local == null) {
                return null;
            }
            return local.toLocalDateTime();
        }
        ctxt.wrongTokenException(jp, JsonToken.START_ARRAY, "expected JSON Array or Number");
        return null;
    }
}

/**
 * @since 1.5
 */
public static class DateMidnightDeserializer
    extends JodaDeserializer<DateMidnight>
{
    public DateMidnightDeserializer() { super(DateMidnight.class); }

    @Override
    public DateMidnight deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        // We'll accept either long (timestamp) or array:
        if (jp.isExpectedStartArrayToken()) {
            jp.nextToken(); // VALUE_NUMBER_INT
            int year = jp.getIntValue();
            jp.nextToken(); // VALUE_NUMBER_INT
            int month = jp.getIntValue();
            jp.nextToken(); // VALUE_NUMBER_INT
            int day = jp.getIntValue();
            if (jp.nextToken() != JsonToken.END_ARRAY) {
                ctxt.wrongTokenException(jp, JsonToken.END_ARRAY, "after DateMidnight ints");
            }
            return new DateMidnight(year, month, day);
        }
        switch (jp.getCurrentToken()) {
        case VALUE_NUMBER_INT:
            return new DateMidnight(jp.getLongValue());
        case VALUE_STRING:
            DateTime local = parseLocal(jp);
            if (local == null) {
```

```
                return null;
            }
            return local.toDateMidnight();
        }
        ctxt.wrongTokenException(jp, JsonToken.START_ARRAY, "expected JSON Array, Number or String");
        return null;
    }
  }
}
/**
Contains extended support for "external" packages: things that
may or may not be present in runtime environment, but that are
commonly enough used so that explicit support can be added.
<p>
Currently supported extensions include:
<ul>
<li>Support for Java 1.5 core XML datatypes: the reason these are
considered "external" is that some platforms that claim to be 1.5 conformant
are only partially so (Google Android, GAE) and do not included these
types.
 </li>
<li>Joda time. This package has superior date/time handling functionality,
and is thus supported. However, to minimize forced dependencies this
support is added as extension so that Joda is not needed by Jackson
itself: but if it is present, its core types are supported to some
degree
 </li>
</ul>

*/

package org.codehaus.jackson.map.ext;
package org.codehaus.jackson.map.ext;

import java.io.IOException;
import java.util.*;

import org.joda.time.*;
import org.joda.time.format.DateTimeFormatter;
import org.joda.time.format.ISODateTimeFormat;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.ser.SerializerBase;
import org.codehaus.jackson.map.util.Provider;

/**
* Provider for serializers that handle some basic data types
```

```
* for <a href="http://joda-time.sourceforge.net/">Joda</a> date/time library.
*<p>
* Since version 1.5, more types are supported. These types use slightly
* different approach to serialization than core date types: "timestamp"
* notation is implemented using JSON arrays, for improved readability.
*
* @since 1.4
*/
public class JodaSerializers
  implements Provider<Map.Entry<Class<?>,JsonSerializer<?>>>
{
  final static HashMap<Class<?>,JsonSerializer<?>> _serializers = new HashMap<Class<?>,JsonSerializer<?>>();
  static {
     _serializers.put(DateTime.class, new DateTimeSerializer());
     _serializers.put(LocalDateTime.class, new LocalDateTimeSerializer());
     _serializers.put(LocalDate.class, new LocalDateSerializer());
     _serializers.put(DateMidnight.class, new DateMidnightSerializer());
  }

  public JodaSerializers() { }

  public Collection<Map.Entry<Class<?>,JsonSerializer<?>>> provide() {
     return _serializers.entrySet();
  }

  /*
  /**********************************************************
  /* Intermediate base classes
  /**********************************************************
   */

  protected abstract static class JodaSerializer<T> extends SerializerBase<T>
  {
     final static DateTimeFormatter _localDateTimeFormat = ISODateTimeFormat.dateTime();
     final static DateTimeFormatter _localDateFormat = ISODateTimeFormat.date();

     protected JodaSerializer(Class<T> cls) { super(cls); }

     protected String printLocalDateTime(ReadablePartial dateValue)
        throws IOException, JsonProcessingException
     {
        return _localDateTimeFormat.print(dateValue);
     }

     protected String printLocalDate(ReadablePartial dateValue)
        throws IOException, JsonProcessingException
     {
        return _localDateFormat.print(dateValue);
```

```
    }

    protected String printLocalDate(ReadableInstant dateValue)
        throws IOException, JsonProcessingException
    {
        return _localDateFormat.print(dateValue);
    }
}

/*
/**********************************************************
/* Concrete serializers
/**********************************************************
 */

public final static class DateTimeSerializer
    extends JodaSerializer<DateTime>
{
    public DateTimeSerializer() { super(DateTime.class); }

    @Override
    public void serialize(DateTime value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        if (provider.isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)) {
            jgen.writeNumber(value.getMillis());
        } else {
            jgen.writeString(value.toString());
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, java.lang.reflect.Type typeHint)
    {
        return
createSchemaNode(provider.isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)
                ? "number" : "string", true);
    }
}

/**
 *
 * @since 1.5
 */
public final static class LocalDateTimeSerializer
    extends JodaSerializer<LocalDateTime>
{
    public LocalDateTimeSerializer() { super(LocalDateTime.class); }
```

```
    @Override
    public void serialize(LocalDateTime dt, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonGenerationException
    {
      if (provider.isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)) {
        // Timestamp here actually means an array of values
        jgen.writeStartArray();
        jgen.writeNumber(dt.year().get());
        jgen.writeNumber(dt.monthOfYear().get());
        jgen.writeNumber(dt.dayOfMonth().get());
        jgen.writeNumber(dt.hourOfDay().get());
        jgen.writeNumber(dt.minuteOfHour().get());
        jgen.writeNumber(dt.secondOfMinute().get());
        jgen.writeNumber(dt.millisOfSecond().get());
        jgen.writeEndArray();
      } else {
        jgen.writeString(printLocalDateTime(dt));
      }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, java.lang.reflect.Type typeHint)
    {
      return
createSchemaNode(provider.isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)
          ? "array" : "string", true);
    }
  }

  public final static class LocalDateSerializer
    extends JodaSerializer<LocalDate>
  {
    public LocalDateSerializer() { super(LocalDate.class); }

    @Override
    public void serialize(LocalDate dt, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonGenerationException
    {
      if (provider.isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)) {
        // Timestamp here actually means an array of values
        jgen.writeStartArray();
        jgen.writeNumber(dt.year().get());
        jgen.writeNumber(dt.monthOfYear().get());
        jgen.writeNumber(dt.dayOfMonth().get());
        jgen.writeEndArray();
      } else {
        jgen.writeString(printLocalDate(dt));
```

```
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, java.lang.reflect.Type typeHint)
    {
        return
createSchemaNode(provider.isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)
                ? "array" : "string", true);
    }
  }

  public final static class DateMidnightSerializer
    extends JodaSerializer<DateMidnight>
  {
    public DateMidnightSerializer() { super(DateMidnight.class); }

    @Override
    public void serialize(DateMidnight dt, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        if (provider.isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)) {
            // same as with other date-only values
            jgen.writeStartArray();
            jgen.writeNumber(dt.year().get());
            jgen.writeNumber(dt.monthOfYear().get());
            jgen.writeNumber(dt.dayOfMonth().get());
            jgen.writeEndArray();
        } else {
            jgen.writeString(printLocalDate(dt));
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, java.lang.reflect.Type typeHint)
    {
        return
createSchemaNode(provider.isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)
                ? "array" : "string", true);
    }
  }

}
package org.codehaus.jackson.map.ext;

import java.io.IOException;
import org.w3c.dom.Node;
import  org.w3c.dom.bootstrap.DOMImplementationRegistry;
```

```
import  org.w3c.dom.ls.DOMImplementationLS;
import  org.w3c.dom.ls.LSSerializer;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.ser.SerializerBase;

public class DOMSerializer
  extends SerializerBase<Node>
{
  protected final DOMImplementationLS _domImpl;

  public DOMSerializer()
  {
     super(Node.class);
     DOMImplementationRegistry registry;
     try {
        registry = DOMImplementationRegistry.newInstance();
     } catch (Exception e) {
        throw new IllegalStateException("Could not instantiate DOMImplementationRegistry: "+e.getMessage(), e);
     }
     _domImpl = (DOMImplementationLS)registry.getDOMImplementation("LS");
  }

  @Override
  public void serialize(Node value, JsonGenerator jgen, SerializerProvider provider)
     throws IOException, JsonGenerationException
  {
     if (_domImpl == null) throw new IllegalStateException("Could not find DOM LS");
     LSSerializer writer = _domImpl.createLSSerializer();
     jgen.writeString(writer.writeToString(value));
  }

  @Override
  public JsonNode getSchema(SerializerProvider provider, java.lang.reflect.Type typeHint)
  {
     // Well... it is serialized as String
     return createSchemaNode("string", true);
  }
}
package org.codehaus.jackson.map.ext;

import java.util.Collection;
import java.util.Map;

import org.codehaus.jackson.map.*;
```

```
import org.codehaus.jackson.map.deser.StdDeserializer;
import org.codehaus.jackson.map.introspect.BasicBeanDescription;
import org.codehaus.jackson.map.util.Provider;
import org.codehaus.jackson.type.JavaType;

/**
* Helper class used for isolating details of handling optional+external types (Joda datetime,
* javax.xml classes) from standard factories that offer them.
*
* @author tatu
*
* @since 1.6.1
*/
public class OptionalHandlerFactory
{
  /* 1.6.1+ To make 2 main "optional" handler groups (javax.xml.stream, Joda date/time)
   * more dynamic, we better only figure out handlers completely dynamically, if and
   * when they are needed. To do this we need to assume package prefixes.
   */

  private final static String PACKAGE_PREFIX_JODA_DATETIME = "org.joda.time.";
  private final static String PACKAGE_PREFIX_JAVAX_XML = "javax.xml.";

  private final static String SERIALIZERS_FOR_JODA_DATETIME =
"org.codehaus.jackson.map.ext.JodaSerializers";
  private final static String SERIALIZERS_FOR_JAVAX_XML =
"org.codehaus.jackson.map.ext.CoreXMLSerializers";
  private final static String DESERIALIZERS_FOR_JODA_DATETIME =
"org.codehaus.jackson.map.ext.JodaDeserializers";
  private final static String DESERIALIZERS_FOR_JAVAX_XML =
"org.codehaus.jackson.map.ext.CoreXMLDeserializers";

  // Plus we also have a single serializer for DOM Node:
  private final static String CLASS_NAME_DOM_NODE = "org.w3c.dom.Node";
  private final static String CLASS_NAME_DOM_DOCUMENT = "org.w3c.dom.Node";
  private final static String SERIALIZER_FOR_DOM_NODE = "org.codehaus.jackson.map.ext.DOMSerializer";
  private final static String DESERIALIZER_FOR_DOM_DOCUMENT =
"org.codehaus.jackson.map.ext.DOMDeserializer$DocumentDeserializer";
  private final static String DESERIALIZER_FOR_DOM_NODE =
"org.codehaus.jackson.map.ext.DOMDeserializer$NodeDeserializer";

  public final static OptionalHandlerFactory instance = new OptionalHandlerFactory();

  protected OptionalHandlerFactory() { }

  /*
  /**********************************************************
  /* Public API
```

```
    /**********************************************************
     */

    public JsonSerializer<?> findSerializer(SerializationConfig config, JavaType type,
            BasicBeanDescription beanInfo, BeanProperty property)
    {
        Class<?> rawType = type.getRawClass();
        String className = rawType.getName();
        String factoryName;

        if (className.startsWith(PACKAGE_PREFIX_JODA_DATETIME)) {
            factoryName = SERIALIZERS_FOR_JODA_DATETIME;
        } else if (className.startsWith(PACKAGE_PREFIX_JAVAX_XML)
                || hasSupertypeStartingWith(rawType, PACKAGE_PREFIX_JAVAX_XML)) {
            factoryName = SERIALIZERS_FOR_JAVAX_XML;
        } else if (doesImplement(rawType, CLASS_NAME_DOM_NODE)) {
            return (JsonSerializer<?>) instantiate(SERIALIZER_FOR_DOM_NODE);
        } else {
            return null;
        }

        Object ob = instantiate(factoryName);
        if (ob == null) { // could warn, if we had logging system (j.u.l?)
            return null;
        }
        @SuppressWarnings("unchecked")
        Provider<Map.Entry<Class<?>,JsonSerializer<?>>> prov =
    (Provider<Map.Entry<Class<?>,JsonSerializer<?>>>) ob;
        Collection<Map.Entry<Class<?>,JsonSerializer<?>>> entries = prov.provide();

        // first, check for exact match (concrete)
        for (Map.Entry<Class<?>,JsonSerializer<?>> entry : entries) {
            if (rawType == entry.getKey()) {
                return entry.getValue();
            }
        }
        // if no match, check super-type match
        for (Map.Entry<Class<?>,JsonSerializer<?>> entry : entries) {
            if (entry.getKey().isAssignableFrom(rawType)) {
                return entry.getValue();
            }
        }
        // but maybe there's just no match to be found?
        return null;
    }

    public JsonDeserializer<?> findDeserializer(JavaType type, DeserializationConfig config, DeserializerProvider p)
    {
```

```
    Class<?> rawType = type.getRawClass();
    String className = rawType.getName();
    String factoryName;

    if (className.startsWith(PACKAGE_PREFIX_JODA_DATETIME)) {
        factoryName = DESERIALIZERS_FOR_JODA_DATETIME;
    } else if (className.startsWith(PACKAGE_PREFIX_JAVAX_XML)
            || hasSupertypeStartingWith(rawType, PACKAGE_PREFIX_JAVAX_XML)) {
        factoryName = DESERIALIZERS_FOR_JAVAX_XML;
    } else if (doesImplement(rawType, CLASS_NAME_DOM_DOCUMENT)) {
        return (JsonDeserializer<?>) instantiate(DESERIALIZER_FOR_DOM_DOCUMENT);
    } else if (doesImplement(rawType, CLASS_NAME_DOM_NODE)) {
        return (JsonDeserializer<?>) instantiate(DESERIALIZER_FOR_DOM_NODE);
    } else {
        return null;
    }
    Object ob = instantiate(factoryName);
    if (ob == null) { // could warn, if we had logging system (j.u.l?)
        return null;
    }
    @SuppressWarnings("unchecked")
    Provider<StdDeserializer<?>> prov = (Provider<StdDeserializer<?>>) ob;
    Collection<StdDeserializer<?>> entries = prov.provide();

    // first, check for exact match (concrete)
    for (StdDeserializer<?> deser : entries) {
        if (rawType == deser.getValueClass()) {
            return deser;
        }
    }
    // if no match, check super-type match
    for (StdDeserializer<?> deser : entries) {
        if (deser.getValueClass().isAssignableFrom(rawType)) {
            return deser;
        }
    }
    // but maybe there's just no match to be found?
    return null;
}

/*
/**********************************************************
/* Internal helper methods
/**********************************************************
 */

private Object instantiate(String className)
{
```

```
    try {
        return Class.forName(className).newInstance();
    }
    catch (LinkageError e) { }
    // too many different kinds to enumerate here:
    catch (Exception e) { }
    return null;
}

private boolean doesImplement(Class<?> actualType, String classNameToImplement)
{
    for (Class<?> type = actualType; type != null; type = type.getSuperclass()) {
        if (type.getName().equals(classNameToImplement)) {
            return true;
        }
        // or maybe one of super-interfaces
        if (hasInterface(type, classNameToImplement)) {
            return true;
        }
    }
    return false;
}

private boolean hasInterface(Class<?> type, String interfaceToImplement)
{
    Class<?>[] interfaces = type.getInterfaces();
    for (Class<?> iface : interfaces) {
        if (iface.getName().equals(interfaceToImplement)) {
            return true;
        }
    }
    // maybe super-interface?
    for (Class<?> iface : interfaces) {
        if (hasInterface(iface, interfaceToImplement)) {
            return true;
        }
    }
    return false;
}

private boolean hasSupertypeStartingWith(Class<?> rawType, String prefix)
{
    // first, superclasses
    for (Class<?> supertype = rawType.getSuperclass(); supertype != null; supertype = supertype.getSuperclass()) {
        if (supertype.getName().startsWith(prefix)) {
            return true;
        }
    }
```

```
        // then interfaces
        for (Class<?> cls = rawType; cls != null; cls = cls.getSuperclass()) {
            if (hasInterfaceStartingWith(cls, prefix)) {
                return true;
            }
        }
        return false;
    }

    private boolean hasInterfaceStartingWith(Class<?> type, String prefix)
    {
        Class<?>[] interfaces = type.getInterfaces();
        for (Class<?> iface : interfaces) {
            if (iface.getName().startsWith(prefix)) {
                return true;
            }
        }
        // maybe super-interface?
        for (Class<?> iface : interfaces) {
            if (hasInterfaceStartingWith(iface, prefix)) {
                return true;
            }
        }
        return false;
    }

}
package org.codehaus.jackson.map.ext;

import java.io.StringReader;

import javax.xml.parsers.DocumentBuilderFactory;

import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.deser.FromStringDeserializer;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.xml.sax.InputSource;

/**
* Base for serializers that allows parsing DOM Documents from JSON Strings.
* Nominal type can be either {@link org.w3c.dom.Node} or
* {@link org.w3c.dom.Document}.
*/
public abstract class DOMDeserializer<T> extends FromStringDeserializer<T>
{
    final static DocumentBuilderFactory _parserFactory;
    static {
```

```
        _parserFactory = DocumentBuilderFactory.newInstance();
        // yup, only cave men do XML without recognizing namespaces...
        _parserFactory.setNamespaceAware(true);
    }

    protected DOMDeserializer(Class<T> cls) { super(cls); }

    @Override
    public abstract T _deserialize(String value, DeserializationContext ctxt);

    protected final Document parse(String value) throws IllegalArgumentException
    {
        try {
            return _parserFactory.newDocumentBuilder().parse(new InputSource(new StringReader(value)));
        } catch (Exception e) {
            throw new IllegalArgumentException("Failed to parse JSON String as XML: "+e.getMessage(), e);
        }
    }

    /*
    /**********************************************************
    /* Concrete deserializers
    /**********************************************************
     */

    public static class NodeDeserializer extends DOMDeserializer<Node>
    {
        public NodeDeserializer() { super(Node.class); }
        @Override
        public Node _deserialize(String value, DeserializationContext ctxt) throws IllegalArgumentException {
            return parse(value);
        }
    }

    public static class DocumentDeserializer extends DOMDeserializer<Document>
    {
        public DocumentDeserializer() { super(Document.class); }
        @Override
        public Document _deserialize(String value, DeserializationContext ctxt) throws IllegalArgumentException {
            return parse(value);
        }
    }
}
package org.codehaus.jackson.map.ext;

import java.io.IOException;
import java.util.*;
```

```
import javax.xml.datatype.DatatypeConfigurationException;
import javax.xml.datatype.DatatypeFactory;
import javax.xml.datatype.Duration;
import javax.xml.datatype.XMLGregorianCalendar;
import javax.xml.namespace.QName;

import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.deser.StdDeserializer;
import org.codehaus.jackson.map.deser.FromStringDeserializer;
import org.codehaus.jackson.map.deser.StdScalarDeserializer;
import org.codehaus.jackson.map.util.Provider;

/**
* Container deserializers that handle "core" XML types: ones included in standard
* JDK 1.5. Types are directly needed by JAXB, and are thus supported within core
* mapper package, not in "xc" package.
*
* @since 1.3
*/
public class CoreXMLDeserializers
  implements Provider<StdDeserializer<?>>
{
  /**
   * Data type factories are thread-safe after instantiation (and
   * configuration, if any); and since instantion (esp. implementation
   * introspection) can be expensive we better reuse the instance.
   */
  final static DatatypeFactory _dataTypeFactory;
  static {
     try {
        _dataTypeFactory = DatatypeFactory.newInstance();
     } catch (DatatypeConfigurationException e) {
        throw new RuntimeException(e);
     }
  }

  /*
  /**********************************************************
  /* Provider implementation
  /**********************************************************
   */

  /**
   * Method called by {@link org.codehaus.jackson.map.deser.BasicDeserializerFactory}
   * to register deserializers this class provides.
   */
```

```
public Collection<StdDeserializer<?>> provide()
{
   return Arrays.asList(new StdDeserializer<?>[] {
      new DurationDeserializer()
      ,new GregorianCalendarDeserializer()
      ,new QNameDeserializer()
   });
}

/*
/**********************************************************
/* Concrete deserializers
/**********************************************************
 */

public static class DurationDeserializer
   extends FromStringDeserializer<Duration>
{
   public DurationDeserializer() { super(Duration.class); }

   @Override
   protected Duration _deserialize(String value, DeserializationContext ctxt)
      throws IllegalArgumentException
   {
      return _dataTypeFactory.newDuration(value);
   }
}

public static class GregorianCalendarDeserializer
   extends StdScalarDeserializer<XMLGregorianCalendar>
{
   public GregorianCalendarDeserializer() { super(XMLGregorianCalendar.class); }

   @Override
   public XMLGregorianCalendar deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException
   {
      Date d = _parseDate(jp, ctxt);
      if (d == null) {
         return null;
      }
      GregorianCalendar calendar = new GregorianCalendar();
      calendar.setTime(d);
      return _dataTypeFactory.newXMLGregorianCalendar(calendar);
   }
}

public static class QNameDeserializer
```

```
    extends FromStringDeserializer<QName>
  {
    public QNameDeserializer() { super(QName.class); }

    @Override
    protected QName _deserialize(String value, DeserializationContext ctxt)
      throws IllegalArgumentException
    {
      return QName.valueOf(value);
    }
  }
}
package org.codehaus.jackson.map.ext;

import java.io.IOException;
import java.lang.reflect.Type;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import javax.xml.datatype.Duration;
import javax.xml.datatype.XMLGregorianCalendar;
import javax.xml.namespace.QName;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.ser.SerializerBase;
import org.codehaus.jackson.map.ser.StdSerializers;
import org.codehaus.jackson.map.ser.ToStringSerializer;
import org.codehaus.jackson.map.util.Provider;

/**
* Provider for serializers of XML types that are part of full JDK 1.5, but
* that some alleged 1.5 platforms are missing (Android, GAE).
* And for this reason these are added using more dynamic mechanism.
*<p>
* Note: since many of classes defined are abstract, caller must take
* care not to just use straight equivalency check but rather consider
* subclassing as well.
*
* @since 1.4
*/
public class CoreXMLSerializers
  implements Provider<Map.Entry<Class<?>,JsonSerializer<?>>>
```

```
{
  final static HashMap<Class<?>,JsonSerializer<?>> _serializers = new HashMap<Class<?>,JsonSerializer<?>>();
  /**
   * We will construct instances statically, during class loading, to try to
   * make things fail-fast, i.e. to catch problems as soon as possible.
   */
  static {
     ToStringSerializer tss = ToStringSerializer.instance;
     _serializers.put(Duration.class, tss);
     _serializers.put(XMLGregorianCalendar.class, new XMLGregorianCalendarSerializer());
     _serializers.put(QName.class, tss);
  }

  public Collection<Map.Entry<Class<?>,JsonSerializer<?>>> provide() {
     return _serializers.entrySet();
  }

  public static class XMLGregorianCalendarSerializer extends SerializerBase<XMLGregorianCalendar>
  {
     public XMLGregorianCalendarSerializer() {
        super(XMLGregorianCalendar.class);
     }

     @Override
     public void serialize(XMLGregorianCalendar value, JsonGenerator jgen, SerializerProvider provider)
           throws IOException, JsonGenerationException {
        StdSerializers.CalendarSerializer.instance.serialize(value.toGregorianCalendar(), jgen, provider);
     }

     @Override
     public JsonNode getSchema(SerializerProvider provider, Type typeHint) throws JsonMappingException {
        return StdSerializers.CalendarSerializer.instance.getSchema(provider, typeHint);
     }
  }
}
package org.codehaus.jackson.map;

import org.codehaus.jackson.Version;
import org.codehaus.jackson.Versioned;
import org.codehaus.jackson.map.deser.BeanDeserializerModifier;
import org.codehaus.jackson.map.ser.BeanSerializerModifier;

/**
* Simple interface for extensions that can be registered with {@link ObjectMapper}
* to provide a well-defined set of extensions to default functionality; such as
* support for new data types.
*
* @since 1.7
```

```
*/
public abstract class Module
  implements Versioned
{
  /*
  /**********************************************************
  /* Simple accessors
  /**********************************************************
   */

  /**
   * Method that returns identifier for module; this can be used by Jackson
   * for informational purposes, as well as in associating extensions with
   * module that provides them.
   */
  public abstract String getModuleName();

  /**
   * Method that returns version of this module. Can be used by Jackson for
   * informational purposes.
   */
  public abstract Version version();

  /*
  /**********************************************************
  /* Life-cycle: registration
  /**********************************************************
   */

  /**
   * Method called by {@link ObjectMapper} when module is registered.
   * It is called to let module register functionality it provides,
   * using callback methods passed-in context object exposes.
   */
  public abstract void setupModule(SetupContext context);

  /*
  /**********************************************************
  /* Helper types
  /**********************************************************
   */

  /**
   * Interface Jackson exposes to modules for purpose of registering
   * extended functionality.
   */
  public interface SetupContext
  {
```

```
/*
/**********************************************************
/* Simple accessors
/**********************************************************
 */

/**
 * Method that returns version information about {@link ObjectMapper}
 * that implements this context. Modules can use this to choose
 * different settings or initialization order; or even decide to fail
 * set up completely if version is compatible with module.
 */
public Version getMapperVersion();

/**
 * Method that returns current deserialization configuration
 * settings. Since modules may be interested in these settings,
 * caller should make sure to make changes to settings before
 * module registrations.
 */
public DeserializationConfig getDeserializationConfig();

/**
 * Method that returns current serialization configuration
 * settings. Since modules may be interested in these settings,
 * caller should make sure to make changes to settings before
 * module registrations.
 *
 * @since 1.7.1 (1.7.0 unfortunately had a typo in method name!)
 */
public SerializationConfig getSerializationConfig();

/**
 * @deprecated Typo in method name included in 1.7.0; with 1.7.1 and
 *    above, please use correctly named {@link #getSerializationConfig()}.
 */
@Deprecated
public SerializationConfig getSeserializationConfig();

/*
/**********************************************************
/* Handler registration
/**********************************************************
 */

/**
 * Method that module can use to register additional deserializers to use for
 * handling types.
```

```
     *
     * @param d Object that can be called to find deserializer for types supported
     *   by module (null returned for non-supported types)
     */
    public void addDeserializers(Deserializers d);

    /**
     * Method that module can use to register additional serializers to use for
     * handling types.
     *
     * @param s Object that can be called to find serializer for types supported
     *   by module (null returned for non-supported types)
     */
    public void addSerializers(Serializers s);

    /**
     * Method that module can use to register additional modifier objects to
     * customize configuration and construction of bean deserializers.
     *
     * @param mod Modifier to register
     */
    public void addBeanDeserializerModifier(BeanDeserializerModifier mod);

    /**
     * Method that module can use to register additional modifier objects to
     * customize configuration and construction of bean serializers.
     *
     * @param mod Modifier to register
     */
    public void addBeanSerializerModifier(BeanSerializerModifier mod);

    /**
     * Method for registering specified {@link AnnotationIntrospector} as the highest
     * priority introspector (will be chained with existing introspector(s) which
     * will be used as fallbacks for cases this introspector does not handle)
     *
     * @param ai Annotation introspector to register.
     */
    public void insertAnnotationIntrospector(AnnotationIntrospector ai);

    /**
     * Method for registering specified {@link AnnotationIntrospector} as the lowest
     * priority introspector, chained with existing introspector(s) and called
     * as fallback for cases not otherwise handled.
     *
     * @param ai Annotation introspector to register.
     */
    public void appendAnnotationIntrospector(AnnotationIntrospector ai);
```

```
        /**
         * Method used for defining mix-in annotations to use for augmenting
         * specified class or interface.
         * All annotations from
         * <code>mixinSource</code> are taken to override annotations
         * that <code>target</code> (or its supertypes) has.
         *<p>
         * Note: mix-ins are registered both for serialization and deserialization
         * (which can be different internally).
         *<p>
         * Note: currently only one set of mix-in annotations can be defined for
         * a single class; so if multiple modules register mix-ins, highest
         * priority one (last one registered) will have priority over other modules.
         *
         * @param target Class (or interface) whose annotations to effectively override
         * @param mixinSource Class (or interface) whose annotations are to
         *   be "added" to target's annotations, overriding as necessary
         */
        public void setMixInAnnotations(Class<?> target, Class<?> mixinSource);
    }
}
/**
* Functionality needed for Bean introspection, required for detecting
* accessors and mutators for Beans, as well as locating and handling
* method annotations.
*<p>
* Beyond collecting annotations, additional "method annotation inheritance"
* is also supported: whereas regular JDK classes do not add annotations
* from overridden methods in any situation. But code in this package does.
* Similarly class-annotations are inherited properly from interfaces, in
* addition to abstract and concrete classes.
*/
package org.codehaus.jackson.map.introspect;
package org.codehaus.jackson.map.introspect;

import java.lang.annotation.Annotation;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.Modifier;
import java.lang.reflect.Type;

import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.map.type.TypeBindings;
import org.codehaus.jackson.map.type.TypeFactory;

/**
* Shared base class used for anything on which annotations (included
* within a {@link AnnotationMap}).
```

```
*/
public abstract class Annotated
{
  protected Annotated() { }

  public abstract <A extends Annotation> A getAnnotation(Class<A> acls);

  public final <A extends Annotation> boolean hasAnnotation(Class<A> acls)
  {
     return getAnnotation(acls) != null;
  }

  /**
   * Method that can be used to find actual JDK element that this instance
   * represents. It is non-null, except for method/constructor parameters
   * which do not have a JDK counterpart.
   */
  public abstract AnnotatedElement getAnnotated();

  protected abstract int getModifiers();

  public final boolean isPublic() {
     return Modifier.isPublic(getModifiers());
  }

  public abstract String getName();

  /**
   * Full generic type of the annotated element; definition
   * of what exactly this means depends on sub-class.
   */
  public JavaType getType(TypeBindings context) {
     return TypeFactory.type(getGenericType(), context);
  }

  /**
   * Full generic type of the annotated element; definition
   * of what exactly this means depends on sub-class.
   *
   * @since 1.5
   */
  public abstract Type getGenericType();

  /**
   * "Raw" type (type-erased class) of the annotated element; definition
   * of what exactly this means depends on sub-class.
   *
   * @since 1.5
```

```
   */
  public abstract Class<?> getRawType();
}
package org.codehaus.jackson.map.introspect;

import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.util.*;

import org.codehaus.jackson.map.AnnotationIntrospector;
import org.codehaus.jackson.map.BeanDescription;
import org.codehaus.jackson.map.annotate.JsonSerialize;
import org.codehaus.jackson.map.introspect.VisibilityChecker;
import org.codehaus.jackson.map.type.TypeBindings;
import org.codehaus.jackson.map.util.Annotations;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.type.JavaType;

/**
* Default {@link BeanDescription} implementation.
* Can theoretically be subclassed to customize
* some aspects of property introspection.
*/
public class BasicBeanDescription extends BeanDescription
{
  /*
  /**********************************************************
  /* Configuration
  /**********************************************************
   */

  /**
   * Information collected about the class introspected.
   */
  final protected AnnotatedClass _classInfo;

  final protected AnnotationIntrospector _annotationIntrospector;

  /**
   * We may need type bindings for the bean type. If so, we'll
   * construct it lazily
   */
  protected TypeBindings _bindings;

  /*
  /**********************************************************
  /* Life-cycle
```

```
/**********************************************************
 */

public BasicBeanDescription(JavaType type, AnnotatedClass ac,
                            AnnotationIntrospector ai)

{
 super(type);
 _classInfo = ac;
    _annotationIntrospector = ai;
}

/*
/**********************************************************
/* Simple accessors from BeanDescription
/**********************************************************
 */

/**
 * Method for checking whether class being described has any
 * annotations recognized by registered annotation introspector.
 */
@Override
public boolean hasKnownClassAnnotations() {
    return _classInfo.hasAnnotations();
}

@Override
public Annotations getClassAnnotations() {
    return _classInfo.getAnnotations();
}

@Override
public TypeBindings bindingsForBeanType()
{
    if (_bindings == null) {
        _bindings = new TypeBindings(_type);
    }
    return _bindings;
}

/*
/**********************************************************
/* Simple accessors, extended
/**********************************************************
 */

public AnnotatedClass getClassInfo() { return _classInfo; }
```

```
    public AnnotatedMethod findMethod(String name, Class<?>[] paramTypes)
    {
        return _classInfo.findMethod(name, paramTypes);
    }

    /**
     * @param fixAccess If true, method is allowed to fix access to the
     *   default constructor (to be able to call non-public constructor);
     *   if false, has to use constructor as is.
     *
     * @return Instance of class represented by this descriptor, if
     *   suitable default constructor was found; null otherwise.
     */
    public Object instantiateBean(boolean fixAccess)
    {
        AnnotatedConstructor ac = _classInfo.getDefaultConstructor();
        if (ac == null) {
            return null;
        }
        if (fixAccess) {
            ac.fixAccess();
        }
        try {
            return ac.getAnnotated().newInstance();
        } catch (Exception e) {
            Throwable t = e;
            while (t.getCause() != null) {
                t = t.getCause();
            }
            if (t instanceof Error) throw (Error) t;
            if (t instanceof RuntimeException) throw (RuntimeException) t;
            throw new IllegalArgumentException("Failed to instantiate bean of type
"+_classInfo.getAnnotated().getName()+": ("+t.getClass().getName()+") "+t.getMessage(), t);
        }
    }

    /*
    /**********************************************************
    /* Basic API
    /**********************************************************
     */

    /*
    /**********************************************************
    /* Introspection for serialization (write JSON), getters
    /**********************************************************
     */
```

```
@Override
public LinkedHashMap<String,AnnotatedMethod> findGetters(VisibilityChecker<?> visibilityChecker,
      Collection<String> ignoredProperties)
{
   LinkedHashMap<String,AnnotatedMethod> results = new LinkedHashMap<String,AnnotatedMethod>();
   for (AnnotatedMethod am : _classInfo.memberMethods()) {
      /* note: signature has already been checked to some degree
       * via filters; however, no checks were done for arg count
       */
      // 16-May-2009, tatu: JsonIgnore processed earlier already
      if (am.getParameterCount() != 0) {
         continue;
      }
      /* So far so good: final check, then; has to either
       * (a) be marked with JsonProperty (/JsonGetter/JsonSerialize) OR
       * (b) be public AND have suitable name (getXxx or isXxx)
       */
      String propName = _annotationIntrospector.findGettablePropertyName(am);
      if (propName != null) {
         /* As per [JACKSON-64], let's still use mangled
          * name if possible; and only if not use unmodified
          * method name
          */
         if (propName.length() == 0) {
            propName = okNameForAnyGetter(am, am.getName());
            if (propName == null) {
               propName = am.getName();
            }
         }
      } else {
         propName = am.getName();
         // [JACKSON-166], need to separate getXxx/isXxx methods
         if (propName.startsWith("get")) { // nope, but is public bean-getter name?
            if (!visibilityChecker.isGetterVisible(am)) {
               continue;
            }
            propName = okNameForGetter(am, propName);
         } else {
            if (!visibilityChecker.isIsGetterVisible(am)) {
               continue;
            }
            propName = okNameForIsGetter(am, propName);
         }
         // null return value means 'not valid'
         if (propName == null) continue;
         // [JACKSON-384] Plus, should not include "AnyGetter" as regular getter..
         if (_annotationIntrospector.hasAnyGetterAnnotation(am)) continue;
```

```
            }

            if (ignoredProperties != null) {
                if (ignoredProperties.contains(propName)) {
                    continue;
                }
            }

            /* Yup, it is a valid name. But now... do we have a conflict?
             * If so, should throw an exception
             */
            AnnotatedMethod old = results.put(propName, am);
            if (old != null) {
                String oldDesc = old.getFullName();
                String newDesc = am.getFullName();
                throw new IllegalArgumentException("Conflicting getter definitions for property \""+propName+"\":
"+oldDesc+" vs "+newDesc);
            }
        }
        return results;
    }

    /**
     * Method for locating the getter method that is annotated with
     * {@link org.codehaus.jackson.annotate.JsonValue} annotation,
     * if any. If multiple ones are found,
     * an error is reported by throwing {@link IllegalArgumentException}
     */
    public AnnotatedMethod findJsonValueMethod()
    {
        AnnotatedMethod found = null;
        for (AnnotatedMethod am : _classInfo.memberMethods()) {
            // must be marked with "JsonValue" (or similar)
            if (!_annotationIntrospector.hasAsValueAnnotation(am)) {
                continue;
            }
            if (found != null) {
                throw new IllegalArgumentException("Multiple methods with active 'as-value' annotation
("+found.getName()+"(), "+am.getName()+")");
            }
            // Also, must have getter signature
            /* 18-May-2009, tatu: Should this be moved to annotation
             *  introspector, to give better error message(s)? For now
             *  will leave here, may want to reconsider in future.
             */
            if (!ClassUtil.hasGetterSignature(am.getAnnotated())) {
                throw new IllegalArgumentException("Method "+am.getName()+"() marked with an 'as-value' annotation,
but does not have valid getter signature (non-static, takes no args, returns a value)");
```

```
        }
        found = am;
    }
    return found;
}

/*
/**********************************************************
/* Introspection for serialization, factories
/**********************************************************
 */

/**
 * Method that will locate the no-arg constructor for this class,
 * if it has one, and that constructor has not been marked as
 * ignorable.
 * Method will also ensure that the constructor is accessible.
 */
public Constructor<?> findDefaultConstructor()
{
    AnnotatedConstructor ac = _classInfo.getDefaultConstructor();
    if (ac == null) {
        return null;
    }
    return ac.getAnnotated();
}

public List<AnnotatedConstructor> getConstructors()
{
    return _classInfo.getConstructors();
}

public List<AnnotatedMethod> getFactoryMethods()
{
    // must filter out anything that clearly is not a factory method
    List<AnnotatedMethod> candidates = _classInfo.getStaticMethods();
    if (candidates.isEmpty()) {
        return candidates;
    }
    ArrayList<AnnotatedMethod> result = new ArrayList<AnnotatedMethod>();
    for (AnnotatedMethod am : candidates) {
        if (isFactoryMethod(am)) {
            result.add(am);
        }
    }
    return result;
}
```

```
/**
 * Method that can be called to locate a single-arg constructor that
 * takes specified exact type (will not accept supertype constructors)
 *
 * @param argTypes Type(s) of the argument that we are looking for
 */
public Constructor<?> findSingleArgConstructor(Class<?>... argTypes)
{
   for (AnnotatedConstructor ac : _classInfo.getConstructors()) {
      // This list is already filtered to only include accessible
      /* (note: for now this is a redundant check; but in future
       * that may change; thus leaving here for now)
       */
      if (ac.getParameterCount() == 1) {
         Class<?> actArg = ac.getParameterClass(0);
         for (Class<?> expArg : argTypes) {
            if (expArg == actArg) {
               return ac.getAnnotated();
            }
         }
      }
   }
   return null;
}

/**
 * Method that can be called to find if introspected class declares
 * a static "valueOf" factory method that returns an instance of
 * introspected type, given one of acceptable types.
 *
 * @param expArgTypes Types that the matching single argument factory
 *   method can take: will also accept super types of these types
 *   (ie. arg just has to be assignable from expArgType)
 */
public Method findFactoryMethod(Class<?>... expArgTypes)
{
   // So, of all single-arg static methods:
   for (AnnotatedMethod am : _classInfo.getStaticMethods()) {
      if (isFactoryMethod(am)) {
         // And must take one of expected arg types (or supertype)
         Class<?> actualArgType = am.getParameterClass(0);
         for (Class<?> expArgType : expArgTypes) {
            // And one that matches what we would pass in
            if (actualArgType.isAssignableFrom(expArgType)) {
               return am.getAnnotated();
            }
         }
      }
```

```
    }
    return null;
}

protected boolean isFactoryMethod(AnnotatedMethod am)
{
    /* First: return type must be compatible with the introspected class
     * (i.e. allowed to be sub-class, although usually is the same
     * class)
     */
    Class<?> rt = am.getRawType();
    if (!getBeanClass().isAssignableFrom(rt)) {
        return false;
    }

    /* Also: must be a recognized factory method, meaning:
     * (a) marked with @JsonCreator annotation, or
     * (a) "valueOf" (at this point, need not be public)
     */
    if (_annotationIntrospector.hasCreatorAnnotation(am)) {
        return true;
    }
    if ("valueOf".equals(am.getName())) {
        return true;
    }
    return false;
}

/**
 * Method for getting ordered list of named Creator properties.
 * Returns an empty list is none found. If multiple Creator
 * methods are defined, order between properties from different
 * methods is undefined; however, properties for each such
 * Creator are ordered properly relative to each other. For the
 * usual case of just a single Creator, named properties are
 * thus properly ordered.
 */
public List<String> findCreatorPropertyNames()
{
    List<String> names = null;

    for (int i = 0; i < 2; ++i) {
        List<? extends AnnotatedWithParams> l = (i == 0)
            ? getConstructors() : getFactoryMethods();
        for (AnnotatedWithParams creator : l) {
            int argCount = creator.getParameterCount();
            if (argCount < 1) continue;
            String name = _annotationIntrospector.findPropertyNameForParam(creator.getParameter(0));
```

```
            if (name == null) continue;
            if (names == null) {
                names = new ArrayList<String>();
            }
            names.add(name);
            for (int p = 1; p < argCount; ++p) {
                names.add(_annotationIntrospector.findPropertyNameForParam(creator.getParameter(p)));
            }
        }
    }
    if (names == null) {
        return Collections.emptyList();
    }
    return names;
}

/*
/**********************************************************
/* Introspection for serialization, fields
/**********************************************************
 */

public LinkedHashMap<String,AnnotatedField> findSerializableFields(VisibilityChecker<?> vchecker,
                                                                   Collection<String> ignoredProperties)
{
    return _findPropertyFields(vchecker, ignoredProperties, true);
}

/*
/**********************************************************
/* Introspection for serialization, other
/**********************************************************
 */

/**
 * Method for determining whether null properties should be written
 * out for a Bean of introspected type. This is based on global
 * feature (lowest priority, passed as argument)
 * and per-class annotation (highest priority).
 */
public JsonSerialize.Inclusion findSerializationInclusion(JsonSerialize.Inclusion defValue)
{
    return _annotationIntrospector.findSerializationInclusion(_classInfo, defValue);
}

/*
/**********************************************************
/* Introspection for deserialization, setters:
```

```
/**********************************************************
 */

@Override
public LinkedHashMap<String,AnnotatedMethod> findSetters(VisibilityChecker<?> vchecker)
{
   LinkedHashMap<String,AnnotatedMethod> results = new LinkedHashMap<String,AnnotatedMethod>();
   for (AnnotatedMethod am : _classInfo.memberMethods()) {
      // note: signature has already been checked via filters

      // Arg count != 1 (JsonIgnore checked earlier)
      if (am.getParameterCount() != 1) {
         continue;
      }

      /* So far so good: final check, then; has to either
       * (a) be marked with JsonProperty (/JsonSetter/JsonDeserialize) OR
       * (b) have suitable name (setXxx) (NOTE: need not be
       *    public, unlike with getters)
       */
      String propName = _annotationIntrospector.findSettablePropertyName(am);
      if (propName != null) { // annotation was found
         /* As per [JACKSON-64], let's still use mangled name if
          * possible; and only if not use unmodified method name
          */
         if (propName.length() == 0) {
            propName = okNameForSetter(am);
            // null means it's not named as a Bean getter; fine, use as is
            if (propName == null) {
               propName = am.getName();
            }
         }
      } else { // nope, but is public bean-setter name?
         if (!vchecker.isSetterVisible(am)) {
            continue;
         }
         propName = okNameForSetter(am);
         if (propName == null) { // null means 'not valid'
            continue;
         }
      }

      /* Yup, it is a valid name. But now... do we have a conflict?
       * If so, should throw an exception
       */
      AnnotatedMethod old = results.put(propName, am);
      if (old != null) {
       /* [JACKSON-255] Only throw exception if they are in same class. Must
```

```
             *   be careful to choose "correct" one; first one should actually
             *   have priority
             *
             */
            if (old.getDeclaringClass() == am.getDeclaringClass()) {
                String oldDesc = old.getFullName();
                String newDesc = am.getFullName();
                throw new IllegalArgumentException("Conflicting setter definitions for property \""+propName+"\":
"+oldDesc+" vs "+newDesc);
            }
            // to put earlier one back
            results.put(propName, old);
           }
       }

       return results;
   }



   /**
    * Method used to locate the method of introspected class that
    * implements {@link org.codehaus.jackson.annotate.JsonAnySetter}. If no such method exists
    * null is returned. If more than one are found, an exception
    * is thrown.
    * Additional checks are also made to see that method signature
    * is acceptable: needs to take 2 arguments, first one String or
    * Object; second any can be any type.
    */
   public AnnotatedMethod findAnySetter()
       throws IllegalArgumentException
   {
       AnnotatedMethod found = null;
       for (AnnotatedMethod am : _classInfo.memberMethods()) {
           if (!_annotationIntrospector.hasAnySetterAnnotation(am)) {
               continue;
           }
           if (found != null) {
               throw new IllegalArgumentException("Multiple methods with 'any-setter' annotation
("+found.getName()+"(), "+am.getName()+")");
           }
           int pcount = am.getParameterCount();
           if (pcount != 2) {
               throw new IllegalArgumentException("Invalid 'any-setter' annotation on method "+am.getName()+"():
takes "+pcount+" parameters, should take 2");
           }
           /* Also, let's be somewhat strict on how field name is to be
            * passed; String, Object make sense, others not
            * so much.
```

```
             */
            /* !!! 18-May-2009, tatu: how about enums? Can add support if
             *  requested; easy enough for devs to add support within
             *  method.
             */
            Class<?> type = am.getParameterClass(0);
            if (type != String.class && type != Object.class) {
                throw new IllegalArgumentException("Invalid 'any-setter' annotation on method "+am.getName()+"(): first
argument not of type String or Object, but "+type.getName());
            }
            found = am;
        }
        return found;
    }

    /**
     * Method used to locate the method of introspected class that
     * implements {@link org.codehaus.jackson.annotate.JsonAnyGetter}.
     * If no such method exists null is returned.
     * If more than one are found, an exception is thrown.
     *
     * @since 1.6
     */
    public AnnotatedMethod findAnyGetter() throws IllegalArgumentException
    {
        AnnotatedMethod found = null;
        for (AnnotatedMethod am : _classInfo.memberMethods()) {
            if (!_annotationIntrospector.hasAnyGetterAnnotation(am)) {
                continue;
            }
            if (found != null) {
                throw new IllegalArgumentException("Multiple methods with 'any-getter' annotation
("+found.getName()+"(), "+am.getName()+")");
            }
            /* For now let's require a Map; in future can add support for other
             * types like perhaps Iterable<Map.Entry>?
             */
            Class<?> type = am.getRawType();
            if (!Map.class.isAssignableFrom(type)) {
                throw new IllegalArgumentException("Invalid 'any-getter' annotation on method "+am.getName()+"():
return type is not instance of java.util.Map");
            }
            found = am;
        }
        return found;
    }

    /**
```

```
     * Method for locating all back-reference properties (setters, fields) bean has
     *
     * @since 1.6
     */
    public Map<String,AnnotatedMember> findBackReferenceProperties()
    {
        HashMap<String,AnnotatedMember> result = null;
        // First, gather setter methods
        for (AnnotatedMethod am : _classInfo.memberMethods()) {
            if (am.getParameterCount() == 1) {
                AnnotationIntrospector.ReferenceProperty prop = _annotationIntrospector.findReferenceType(am);
                if (prop != null && prop.isBackReference()) {
                    if (result == null) {
                        result = new HashMap<String,AnnotatedMember>();
                    }
                    if (result.put(prop.getName(), am) != null) {
                        throw new IllegalArgumentException("Multiple back-reference properties with name
'"+prop.getName()+"'");
                    }
                }
            }
        }
        // then settable fields
        for (AnnotatedField af : _classInfo.fields()) {
            AnnotationIntrospector.ReferenceProperty prop = _annotationIntrospector.findReferenceType(af);
            if (prop != null && prop.isBackReference()) {
                if (result == null) {
                    result = new HashMap<String,AnnotatedMember>();
                }
                if (result.put(prop.getName(), af) != null) {
                    throw new IllegalArgumentException("Multiple back-reference properties with name
'"+prop.getName()+"'");
                }
            }
        }
        return result;
    }

    /*
    /**********************************************************
    /* Introspection for deserialization, fields:
    /**********************************************************
     */

    public LinkedHashMap<String,AnnotatedField> findDeserializableFields(VisibilityChecker<?> vchecker,
                                                 Collection<String> ignoredProperties)
    {
        return _findPropertyFields(vchecker, ignoredProperties, false);
```

```
}

/*
/**********************************************************
/* Helper methods for getters
/**********************************************************
 */

public String okNameForAnyGetter(AnnotatedMethod am, String name)
{
   String str = okNameForIsGetter(am, name);
   if (str == null) {
      str = okNameForGetter(am, name);
   }
   return str;
}

public String okNameForGetter(AnnotatedMethod am, String name)
{
   if (name.startsWith("get")) {
      /* 16-Feb-2009, tatu: To handle [JACKSON-53], need to block
       *   CGLib-provided method "getCallbacks". Not sure of exact
       *   safe critieria to get decent coverage without false matches;
       *   but for now let's assume there's no reason to use any
       *   such getter from CGLib.
       *   But let's try this approach...
       */
      if ("getCallbacks".equals(name)) {
         if (isCglibGetCallbacks(am)) {
            return null;
         }
      } else if ("getMetaClass".equals(name)) {
         /* 30-Apr-2009, tatu: [JACKSON-103], need to suppress
          *    serialization of a cyclic (and useless) reference
          */
         if (isGroovyMetaClassGetter(am)) {
            return null;
         }
      }
      return mangleGetterName(am, name.substring(3));
   }
   return null;
}

public String okNameForIsGetter(AnnotatedMethod am, String name)
{
   if (name.startsWith("is")) {
      // plus, must return boolean...
```

```
        Class<?> rt = am.getRawType();
        if (rt != Boolean.class && rt != Boolean.TYPE) {
            return null;
        }
        return mangleGetterName(am, name.substring(2));
    }
    // no, not a match by name
    return null;
}

/**
 * @return Null to indicate that method is not a valid accessor;
 *   otherwise name of the property it is accessor for
 */
protected String mangleGetterName(Annotated a, String basename)
{
    return manglePropertyName(basename);
}

/**
 * This method was added to address [JACKSON-53]: need to weed out
 * CGLib-injected "getCallbacks".
 * At this point caller has detected a potential getter method
 * with name "getCallbacks" and we need to determine if it is
 * indeed injectect by Cglib. We do this by verifying that the
 * result type is "net.sf.cglib.proxy.Callback[]"
 *<p>
 * Also, see [JACKSON-177]; Hibernate may repackage cglib
 * it uses, so we better catch that too
 */
protected boolean isCglibGetCallbacks(AnnotatedMethod am)
{
    Class<?> rt = am.getRawType();
    // Ok, first: must return an array type
    if (rt == null || !rt.isArray()) {
        return false;
    }
    /* And that type needs to be "net.sf.cglib.proxy.Callback".
     * Theoretically could just be a type that implements it, but
     * for now let's keep things simple, fix if need be.
     */
    Class<?> compType = rt.getComponentType();
    // Actually, let's just verify it's a "net.sf.cglib.*" class/interface
    Package pkg = compType.getPackage();
    if (pkg != null) {
        String pname = pkg.getName();
        if (pname.startsWith("net.sf.cglib")
            // also, as per [JACKSON-177]
```

```
            || pname.startsWith("org.hibernate.repackage.cglib")) {
            return true;
        }
    }
    return false;
}

/**
 * Similar to {@link #isCglibGetCallbacks}, need to suppress
 * a cyclic reference to resolve [JACKSON-103]
 */
protected boolean isGroovyMetaClassSetter(AnnotatedMethod am)
{
    Class<?> argType = am.getParameterClass(0);
    Package pkg = argType.getPackage();
    if (pkg != null && pkg.getName().startsWith("groovy.lang")) {
        return true;
    }
    return false;
}

/**
 * Another helper method to deal with rest of [JACKSON-103]
 */
protected boolean isGroovyMetaClassGetter(AnnotatedMethod am)
{
    Class<?> rt = am.getRawType();
    if (rt == null || rt.isArray()) {
        return false;
    }
    Package pkg = rt.getPackage();
    if (pkg != null && pkg.getName().startsWith("groovy.lang")) {
        return true;
    }
    return false;
}

/*
/**********************************************************
/* Helper methods for setters
/**********************************************************
 */

public String okNameForSetter(AnnotatedMethod am)
{
    String name = am.getName();

    /* For mutators, let's not require it to be public. Just need
```

```
     * to be able to call it, i.e. do need to 'fix' access if so
     * (which is done at a later point as needed)
     */
    if (name.startsWith("set")) {
        name = mangleSetterName(am, name.substring(3));
        if (name == null) { // plain old "set" is no good...
            return null;
        }
        if ("metaClass".equals(name)) {
            // 26-Nov-2009 [JACSON-103], need to suppress this internal groovy method
            if (isGroovyMetaClassSetter(am)) {
                return null;
            }
        }
        return name;
    }
    return null;
}

/**
 * @return Null to indicate that method is not a valid accessor;
 *   otherwise name of the property it is accessor for
 */
protected String mangleSetterName(Annotated a, String basename)
{
    return manglePropertyName(basename);
}

/*
/**********************************************************
/* Helper methods for field introspection
/**********************************************************
 */

/**
 * @param vchecker (optional) Object that determines whether specific fields
 *   have enough visibility to be considered for inclusion; if null,
 *   auto-detection is disabled
 * @param ignoredProperties (optional) names of properties to ignore;
 *   any fields that would be recognized as one of these properties
 *   is ignored.
 * @param forSerialization If true, will collect serializable property
 *    fields; if false, deserializable
 *
 * @return Ordered Map with logical property name as key, and
 *    matching field as value.
 */
public LinkedHashMap<String,AnnotatedField> _findPropertyFields(VisibilityChecker<?> vchecker,
```

```
                                    Collection<String> ignoredProperties,
                                    boolean forSerialization)
{
    LinkedHashMap<String,AnnotatedField> results = new LinkedHashMap<String,AnnotatedField>();
    for (AnnotatedField af : _classInfo.fields()) {
        /* note: some pre-filtering has been; no static or transient fields
         * included; nor anything marked as ignorable (@JsonIgnore).
         * Field masking has also been resolved, but it is still possible
         * to get conflicts due to logical name overwrites.
         */

        /* So far so good: final check, then; has to either
         * (a) be marked with JsonProperty (or JsonSerialize) OR
         * (b) be public
         */
        String propName = forSerialization
            ? _annotationIntrospector.findSerializablePropertyName(af)
            : _annotationIntrospector.findDeserializablePropertyName(af)
            ;
        if (propName != null) { // is annotated
            if (propName.length() == 0) {
                propName = af.getName();
            }
        } else { // nope, but may be visible (usually, public, can be recofingured)
            if (!vchecker.isFieldVisible(af)) {
                continue;
            }
            propName = af.getName();
        }

        if (ignoredProperties != null) {
            if (ignoredProperties.contains(propName)) {
                continue;
            }
        }

        /* Yup, it is a valid name. But do we have a conflict?
         * Shouldn't usually happen, but it is possible... and for
         * now let's consider it a problem
         */
        AnnotatedField old = results.put(propName, af);
        if (old != null) {
            /* 21-Feb-2010, tatus: Not necessarily a conflict, still; only
             *    conflict if these are declared in same class (in future, might
             *    not even be conflict then, if types are different? That would
             *    allow "union" types. But for now, let's consider that illegale
             *    to keep things simple.
             */
```

```
            /* Note: we assume that fields are ordered from "oldest" (super-class) to
             * "newest" (sub-classes); this should guarantee proper ordering
             */
            if (old.getDeclaringClass() == af.getDeclaringClass()) {
                String oldDesc = old.getFullName();
                String newDesc = af.getFullName();
                throw new IllegalArgumentException("Multiple fields representing property \""+propName+"\":
"+oldDesc+" vs "+newDesc);
            }
        }
    }
    return results;
  }

  /*
  /**********************************************************
  /* Property name mangling (getFoo -> foo)
  /**********************************************************
   */

  /**
   * Method called to figure out name of the property, given
   * corresponding suggested name based on a method or field name.
   *
   * @param basename Name of accessor/mutator method, not including prefix
   *  ("get"/"is"/"set")
   */
  public static String manglePropertyName(String basename)
  {
    int len = basename.length();

    // First things first: empty basename is no good
    if (len == 0) {
      return null;
    }
    // otherwise, lower case initial chars
    StringBuilder sb = null;
    for (int i = 0; i < len; ++i) {
      char upper = basename.charAt(i);
      char lower = Character.toLowerCase(upper);
      if (upper == lower) {
        break;
      }
      if (sb == null) {
        sb = new StringBuilder(basename);
      }
      sb.setCharAt(i, lower);
    }
```

```
        return (sb == null) ? basename : sb.toString();
    }

    /**
     * Helper method used to describe an annotated element of type
     * {@link Class} or {@link Method}.
     */
    public static String descFor(AnnotatedElement elem)
    {
        if (elem instanceof Class<?>) {
            return "class "+((Class<?>) elem).getName();
        }
        if (elem instanceof Method) {
            Method m = (Method) elem;
            return "method "+m.getName()+" (from class "+m.getDeclaringClass().getName()+")";
        }
        if (elem instanceof Constructor<?>) {
            Constructor<?> ctor = (Constructor<?>) elem;
            // should indicate number of args?
            return "constructor() (from class "+ctor.getDeclaringClass().getName()+")";
        }
        // what else?
        return "unknown type ["+elem.getClass()+"]";
    }
}
package org.codehaus.jackson.map.introspect;

import java.lang.annotation.Annotation;
import java.lang.reflect.*;
import java.util.*;

import org.codehaus.jackson.map.AnnotationIntrospector;
import org.codehaus.jackson.map.ClassIntrospector.MixInResolver;
import org.codehaus.jackson.map.util.Annotations;
import org.codehaus.jackson.map.util.ArrayBuilders;
import org.codehaus.jackson.map.util.ClassUtil;

public final class AnnotatedClass
    extends Annotated
{
    /*
    /**********************************************************
    /* Configuration
    /**********************************************************
     */

    /**
     * Class for which annotations apply, and that owns other
```

```
 * components (constructors, methods)
 */
final protected Class<?> _class;

/**
 * Ordered set of super classes and interfaces of the
 * class itself: included in order of precedence
 */
final protected Collection<Class<?>> _superTypes;

/**
 * Filter used to determine which annotations to gather; used
 * to optimize things so that unnecessary annotations are
 * ignored.
 */
final protected AnnotationIntrospector _annotationIntrospector;

/**
 * Object that knows mapping of mix-in classes (ones that contain
 * annotations to add) with their target classes (ones that
 * get these additional annotations "mixed in").
 */
final protected MixInResolver _mixInResolver;

/**
 * Primary mix-in class; one to use for the annotated class
 * itself. Can be null.
 */
final protected Class<?> _primaryMixIn;

/*
/**********************************************************
/* Gathered information
/**********************************************************
 */

/**
 * Combined list of Jackson annotations that the class has,
 * including inheritable ones from super classes and interfaces
 */
protected AnnotationMap _classAnnotations;

/**
 * Default constructor of the annotated class, if it has one.
 */
protected AnnotatedConstructor _defaultConstructor;

/**
```

```
     * Single argument constructors the class has, if any.
     */
    protected List<AnnotatedConstructor> _constructors;

    /**
     * Single argument static methods that might be usable
     * as factory methods
     */
    protected List<AnnotatedMethod> _creatorMethods;

    /**
     * Member methods of interest; for now ones with 0 or 1 arguments
     * (just optimization, since others won't be used now)
     */
    protected AnnotatedMethodMap  _memberMethods;

    /**
     * Member fields of interest: ones that are either public,
     * or have at least one annotation.
     */
    protected List<AnnotatedField> _fields;

    // // // Lists of explicitly ignored entries (optionally populated)

    /**
     * Optionally populated list that contains member methods that were
     * excluded from applicable methods due to explicit ignore annotation
     */
    protected List<AnnotatedMethod> _ignoredMethods;

    /**
     * Optionally populated list that contains fields that were
     * excluded from applicable fields due to explicit ignore annotation
     */
    protected List<AnnotatedField> _ignoredFields;

    /*
    /**********************************************************
    /* Life-cycle
    /**********************************************************
     */

    /**
     * Constructor will not do any initializations, to allow for
     * configuring instances differently depending on use cases
     */
    private AnnotatedClass(Class<?> cls, List<Class<?>> superTypes,
            AnnotationIntrospector aintr, MixInResolver mir)
```

```
{
    _class = cls;
    _superTypes = superTypes;
    _annotationIntrospector = aintr;
    _mixInResolver = mir;
    _primaryMixIn = (_mixInResolver == null) ? null
        : _mixInResolver.findMixInClassFor(_class);
}

/**
 * Factory method that instantiates an instance. Returned instance
 * will only be initialized with class annotations, but not with
 * any method information.
 */
public static AnnotatedClass construct(Class<?> cls,
        AnnotationIntrospector aintr, MixInResolver mir)
{
    List<Class<?>> st = ClassUtil.findSuperTypes(cls, null);
    AnnotatedClass ac = new AnnotatedClass(cls, st, aintr, mir);
    ac.resolveClassAnnotations();
    return ac;
}

/**
 * Method similar to {@link #construct}, but that will NOT include
 * information from supertypes; only class itself and any direct
 * mix-ins it may have.
 */
public static AnnotatedClass constructWithoutSuperTypes(Class<?> cls,
        AnnotationIntrospector aintr, MixInResolver mir)
{
    List<Class<?>> empty = Collections.emptyList();
    AnnotatedClass ac = new AnnotatedClass(cls, empty, aintr, mir);
    ac.resolveClassAnnotations();
    return ac;
}

/*
/**********************************************************
/* Annotated impl
/**********************************************************
 */

@Override
public Class<?> getAnnotated() { return _class; }

@Override
public int getModifiers() { return _class.getModifiers(); }
```

```
@Override
public String getName() { return _class.getName(); }

@Override
public <A extends Annotation> A getAnnotation(Class<A> acls)
{
   if (_classAnnotations == null) {
      return null;
   }
   return _classAnnotations.get(acls);
}

@Override
public Type getGenericType() {
   return _class;
}

@Override
public Class<?> getRawType() {
   return _class;
}

/*
/**********************************************************
/* Public API, generic accessors
/**********************************************************
 */

/**
 * @return 1.7
 */
public Annotations getAnnotations() { return _classAnnotations; }

public boolean hasAnnotations() { return _classAnnotations.size() > 0; }

public AnnotatedConstructor getDefaultConstructor() { return _defaultConstructor; }

public List<AnnotatedConstructor> getConstructors()
{
   if (_constructors == null) {
      return Collections.emptyList();
   }
   return _constructors;
}

public List<AnnotatedMethod> getStaticMethods()
{
```

```
    if (_creatorMethods == null) {
        return Collections.emptyList();
    }
    return _creatorMethods;
}

public Iterable<AnnotatedMethod> memberMethods()
{
    return _memberMethods;
}

public Iterable<AnnotatedMethod> ignoredMemberMethods()
{
    if (_ignoredMethods == null) {
        List<AnnotatedMethod> l = Collections.emptyList();
        return l;
    }
    return _ignoredMethods;
}

public int getMemberMethodCount()
{
    return _memberMethods.size();
}

public AnnotatedMethod findMethod(String name, Class<?>[] paramTypes)
{
    return _memberMethods.find(name, paramTypes);
}

public int getFieldCount() {
    return (_fields == null) ? 0 : _fields.size();
}

public Iterable<AnnotatedField> fields()
{
    if (_fields == null) {
        List<AnnotatedField> l = Collections.emptyList();
        return l;
    }
    return _fields;
}

public Iterable<AnnotatedField> ignoredFields()
{
    if (_ignoredFields == null) {
        List<AnnotatedField> l = Collections.emptyList();
        return l;
```

```
    }
    return _ignoredFields;
}

/*
/**********************************************************
/* Methods for resolving class annotations
/* (resolution consisting of inheritance, overrides,
/* and injection of mix-ins as necessary)
/**********************************************************
 */

/**
 * Initialization method that will recursively collect Jackson
 * annotations for this class and all super classes and
 * interfaces.
 *<p>
 * Starting with 1.2, it will also apply mix-in annotations,
 * as per [JACKSON-76]
 */
protected void resolveClassAnnotations()
{
    _classAnnotations = new AnnotationMap();
    // add mix-in annotations first (overrides)
    if (_primaryMixIn != null) {
        _addClassMixIns(_classAnnotations, _class, _primaryMixIn);
    }
    // first, annotations from the class itself:
    for (Annotation a : _class.getDeclaredAnnotations()) {
        if (_annotationIntrospector.isHandled(a)) {
            _classAnnotations.addIfNotPresent(a);
        }
    }

    // and then from super types
    for (Class<?> cls : _superTypes) {
        // and mix mix-in annotations in-between
        _addClassMixIns(_classAnnotations, cls);
        for (Annotation a : cls.getDeclaredAnnotations()) {
            if (_annotationIntrospector.isHandled(a)) {
                _classAnnotations.addIfNotPresent(a);
            }
        }
    }

    /* and finally... any annotations there might be for plain
     * old Object.class: separate because for all other purposes
     * it is just ignored (not included in super types)
```

```
     */
    /* 12-Jul-2009, tatu: Should this be done for interfaces too?
     *   For now, yes, seems useful for some cases, and not harmful
     *   for any?
     */
    _addClassMixIns(_classAnnotations, Object.class);
}

/**
 * Helper method for adding any mix-in annotations specified
 * class might have.
 */
protected void _addClassMixIns(AnnotationMap annotations, Class<?> toMask)
{
    if (_mixInResolver != null) {
        _addClassMixIns(annotations, toMask, _mixInResolver.findMixInClassFor(toMask));
    }
}

protected void _addClassMixIns(AnnotationMap annotations, Class<?> toMask,
                               Class<?> mixin)
{
    if (mixin == null) {
        return;
    }
    // Ok, first: annotations from mix-in class itself:
    for (Annotation a : mixin.getDeclaredAnnotations()) {
        if (_annotationIntrospector.isHandled(a)) {
            annotations.addIfNotPresent(a);
        }
    }
    /* And then from its supertypes, if any. But note that we will
     *  only consider super-types up until reaching the masked
     * class (if found); this because often mix-in class
     * is a sub-class (for convenience reasons). And if so, we
     * absolutely must NOT include super types of masked class,
     * as that would inverse precedence of annotations.
     */
    for (Class<?> parent : ClassUtil.findSuperTypes(mixin, toMask)) {
        for (Annotation a : parent.getDeclaredAnnotations()) {
            if (_annotationIntrospector.isHandled(a)) {
                annotations.addIfNotPresent(a);
            }
        }
    }
}

/*
```

```
/**********************************************************
/* Methods for populating creator (ctor, factory) information
/**********************************************************
 */

/**
 * Initialization method that will find out all constructors
 * and potential static factory methods the class has.
 *<p>
 * Starting with 1.2, it will also apply mix-in annotations,
 * as per [JACKSON-76]
 *
 * @param includeAll If true, includes all creator methods; if false,
 *   will only include the no-arguments "default" constructor
 */
public void resolveCreators(boolean includeAll)
{
    // Then see which constructors we have
    _constructors = null;
    for (Constructor<?> ctor : _class.getDeclaredConstructors()) {
        switch (ctor.getParameterTypes().length) {
        case 0:
            _defaultConstructor = _constructConstructor(ctor, true);
            break;
        default:
            if (includeAll) {
                if (_constructors == null) {
                    _constructors = new ArrayList<AnnotatedConstructor>();
                }
                _constructors.add(_constructConstructor(ctor, false));
            }
        }
    }
    // and if need be, augment with mix-ins
    if (_primaryMixIn != null) {
        if (_defaultConstructor != null || _constructors != null) {
            _addConstructorMixIns(_primaryMixIn);
        }
    }


    /* And then... let's remove all constructors that are
     * deemed to be ignorable after all annotations have been
     * properly collapsed.
     */
    if (_defaultConstructor != null) {
        if (_annotationIntrospector.isIgnorableConstructor(_defaultConstructor)) {
            _defaultConstructor = null;
```

```
            }
        }
        if (_constructors != null) {
            // count down to allow safe removal
            for (int i = _constructors.size(); --i >= 0; ) {
                if (_annotationIntrospector.isIgnorableConstructor(_constructors.get(i))) {
                    _constructors.remove(i);
                }
            }
        }

        _creatorMethods = null;

        if (includeAll) {
            /* Then static methods which are potential factory
             * methods
             */
            for (Method m : _class.getDeclaredMethods()) {
                if (!Modifier.isStatic(m.getModifiers())) {
                    continue;
                }
                int argCount = m.getParameterTypes().length;
                // factory methods take at least one arg:
                if (argCount < 1) {
                    continue;
                }
                if (_creatorMethods == null) {
                    _creatorMethods = new ArrayList<AnnotatedMethod>();
                }
                _creatorMethods.add(_constructCreatorMethod(m));
            }
            // mix-ins to mix in?
            if (_primaryMixIn != null && _creatorMethods != null) {
                _addFactoryMixIns(_primaryMixIn);
            }
            // anything to ignore at this point?
            if (_creatorMethods != null) {
                // count down to allow safe removal
                for (int i = _creatorMethods.size(); --i >= 0; ) {
                    if (_annotationIntrospector.isIgnorableMethod(_creatorMethods.get(i))) {
                        _creatorMethods.remove(i);
                    }
                }
            }
        }
    }

    protected void _addConstructorMixIns(Class<?> mixin)
```

```
{
    MemberKey[] ctorKeys = null;
    int ctorCount = (_constructors == null) ? 0 : _constructors.size();
    for (Constructor<?> ctor : mixin.getDeclaredConstructors()) {
        switch (ctor.getParameterTypes().length) {
        case 0:
            if (_defaultConstructor != null) {
                _addMixOvers(ctor, _defaultConstructor, false);
            }
            break;
        default:
            if (ctorKeys == null) {
                ctorKeys = new MemberKey[ctorCount];
                for (int i = 0; i < ctorCount; ++i) {
                    ctorKeys[i] = new MemberKey(_constructors.get(i).getAnnotated());
                }
            }
            MemberKey key = new MemberKey(ctor);

            for (int i = 0; i < ctorCount; ++i) {
                if (!key.equals(ctorKeys[i])) {
                    continue;
                }
                _addMixOvers(ctor, _constructors.get(i), true);
                break;
            }
        }
    }
}

protected void _addFactoryMixIns(Class<?> mixin)
{
    MemberKey[] methodKeys = null;
    int methodCount = _creatorMethods.size();

    for (Method m : mixin.getDeclaredMethods()) {
        if (!Modifier.isStatic(m.getModifiers())) {
            continue;
        }
        if (m.getParameterTypes().length == 0) {
            continue;
        }
        if (methodKeys == null) {
            methodKeys = new MemberKey[methodCount];
            for (int i = 0; i < methodCount; ++i) {
                methodKeys[i] = new MemberKey(_creatorMethods.get(i).getAnnotated());
            }
        }
```

```
            MemberKey key = new MemberKey(m);
            for (int i = 0; i < methodCount; ++i) {
                if (!key.equals(methodKeys[i])) {
                    continue;
                }
                _addMixOvers(m, _creatorMethods.get(i), true);
                break;
            }
        }
    }

    /*
    /**********************************************************
    /* Methods for populating method information
    /**********************************************************
     */

    /**
     * Method for resolving member method information: aggregating all non-static methods
     * and combining annotations (to implement method-annotation inheritance)
     *
     * @param collectIgnored Whether to collect list of ignored methods for later retrieval
     */
    public void resolveMemberMethods(MethodFilter methodFilter, boolean collectIgnored)
    {
        _memberMethods = new AnnotatedMethodMap();
        AnnotatedMethodMap mixins = new AnnotatedMethodMap();
        // first: methods from the class itself
        _addMemberMethods(_class, methodFilter, _memberMethods, _primaryMixIn, mixins);

        // and then augment these with annotations from super-types:
        for (Class<?> cls : _superTypes) {
            Class<?> mixin = (_mixInResolver == null) ? null : _mixInResolver.findMixInClassFor(cls);
            _addMemberMethods(cls, methodFilter, _memberMethods, mixin, mixins);
        }
        // Special case: mix-ins for Object.class? (to apply to ALL classes)
        if (_mixInResolver != null) {
            Class<?> mixin = _mixInResolver.findMixInClassFor(Object.class);
            if (mixin != null) {
                _addMethodMixIns(methodFilter, _memberMethods, mixin, mixins);
            }
        }

        /* Any unmatched mix-ins? Most likely error cases (not matching
         * any method); but there is one possible real use case:
         * exposing Object#hashCode (alas, Object#getClass can NOT be
         * exposed, see [JACKSON-140])
         */
```

```
    if (!mixins.isEmpty()) {
        Iterator<AnnotatedMethod> it = mixins.iterator();
        while (it.hasNext()) {
            AnnotatedMethod mixIn = it.next();
            try {
                Method m = Object.class.getDeclaredMethod(mixIn.getName(), mixIn.getParameterClasses());
                if (m != null) {
                    AnnotatedMethod am = _constructMethod(m);
                    _addMixOvers(mixIn.getAnnotated(), am, false);
                    _memberMethods.add(am);
                }
            } catch (Exception e) { }
        }
    }

    /* And last but not least: let's remove all methods that are
     * deemed to be ignorable after all annotations have been
     * properly collapsed.
     */
    Iterator<AnnotatedMethod> it = _memberMethods.iterator();
    while (it.hasNext()) {
        AnnotatedMethod am = it.next();
        if (_annotationIntrospector.isIgnorableMethod(am)) {
            it.remove();
            if (collectIgnored) {
                _ignoredMethods = ArrayBuilders.addToList(_ignoredMethods, am);
            }
        }
    }
}

protected void _addMemberMethods(Class<?> cls,
        MethodFilter methodFilter, AnnotatedMethodMap methods,
        Class<?> mixInCls, AnnotatedMethodMap mixIns)
{
    // first, mixIns, since they have higher priority then class methods
    if (mixInCls != null) {
        _addMethodMixIns(methodFilter, methods, mixInCls, mixIns);
    }

    if (cls == null) { // just so caller need not check when passing super-class
        return;
    }
    // then methods from the class itself
    for (Method m : cls.getDeclaredMethods()) {
        if (!_isIncludableMethod(m, methodFilter)) {
            continue;
        }
```

```
        AnnotatedMethod old = methods.find(m);
        if (old == null) {
            AnnotatedMethod newM = _constructMethod(m);
            methods.add(newM);
            // Ok, but is there a mix-in to connect now?
            old = mixIns.remove(m);
            if (old != null) {
                _addMixOvers(old.getAnnotated(), newM, false);
            }
        } else {
            /* If sub-class already has the method, we only want to augment
             * annotations with entries that are not masked by sub-class.
             */
            _addMixUnders(m, old);

            /* 06-Jan-2010, tatu: [JACKSON-450] Except that if method we saw first is
             *   from an interface, and we now find a non-interface definition, we should
             *   use this method, but with combination of annotations.
             *   This helps (or rather, is essential) with JAXB annotations and
             *   may also result in faster method calls (interface calls are slightly
             *   costlier than regular method calls)
             */
            if (old.getDeclaringClass().isInterface() && !m.getDeclaringClass().isInterface()) {
                methods.add(old.withMethod(m));
            }
        }
    }
}

protected void _addMethodMixIns(MethodFilter methodFilter, AnnotatedMethodMap methods,
        Class<?> mixInCls, AnnotatedMethodMap mixIns)
{
    for (Method m : mixInCls.getDeclaredMethods()) {
        if (!_isIncludableMethod(m, methodFilter)) {
            continue;
        }
        AnnotatedMethod am = methods.find(m);
        /* Do we already have a method to augment (from sub-class
         * that will mask this mixIn)? If so, add if visible
         * without masking (no such annotation)
         */
        if (am != null) {
            _addMixUnders(m, am);
            /* Otherwise will have precedence, but must wait
             * until we find the real method (mixIn methods are
             * just placeholder, can't be called)
             */
        } else {
```

```
            mixIns.add(_constructMethod(m));
        }
    }
}

/*
/**********************************************************
/* Methods for populating field information
/**********************************************************
 */

/**
 * Method that will collect all member (non-static) fields
 * that are either public, or have at least a single annotation
 * associated with them.
 *
 * @param collectIgnored Whether to collect list of ignored methods for later retrieval
 */
public void resolveFields(boolean collectIgnored)
{
    LinkedHashMap<String,AnnotatedField> foundFields = new LinkedHashMap<String,AnnotatedField>();
    _addFields(foundFields, _class);

    /* And last but not least: let's remove all fields that are
     * deemed to be ignorable after all annotations have been
     * properly collapsed.
     */
    Iterator<Map.Entry<String,AnnotatedField>> it = foundFields.entrySet().iterator();
    while (it.hasNext()) {
        AnnotatedField f = it.next().getValue();
        if (_annotationIntrospector.isIgnorableField(f)) {
            it.remove();
            if (collectIgnored) {
                _ignoredFields = ArrayBuilders.addToList(_ignoredFields, f);
            }
        } else {

        }
    }
    if (foundFields.isEmpty()) {
        _fields = Collections.emptyList();
    } else {
        _fields = new ArrayList<AnnotatedField>(foundFields.size());
        _fields.addAll(foundFields.values());
    }
}

protected void _addFields(Map<String,AnnotatedField> fields, Class<?> c)
```

```
{
    /* First, a quick test: we only care for regular classes (not
     * interfaces, primitive types etc), except for Object.class.
     * A simple check to rule out other cases is to see if there
     * is a super class or not.
     */
    Class<?> parent = c.getSuperclass();
    if (parent != null) {
        // Let's add super-class' fields first, then ours.
        /* 21-Feb-2010, tatu: Need to handle masking: as per [JACKSON-226]
         *    we otherwise get into trouble...
         */
        _addFields(fields, parent);
        for (Field f : c.getDeclaredFields()) {
            // static fields not included, nor transient
            if (!_isIncludableField(f)) {
                continue;
            }
            /* Ok now: we can (and need) not filter out ignorable fields
             * at this point; partly because mix-ins haven't been
             * added, and partly because logic can be done when
             * determining get/settability of the field.
             */
            fields.put(f.getName(), _constructField(f));
        }
        // And then... any mix-in overrides?
        if (_mixInResolver != null) {
            Class<?> mixin = _mixInResolver.findMixInClassFor(c);
            if (mixin != null) {
                _addFieldMixIns(mixin, fields);
            }
        }
    }
}

/**
 * Method called to add field mix-ins from given mix-in class (and its fields)
 * into already collected actual fields (from introspected classes and their
 * super-classes)
 */
protected void _addFieldMixIns(Class<?> mixin, Map<String,AnnotatedField> fields)
{
    for (Field mixinField : mixin.getDeclaredFields()) {
        /* there are some dummy things (static, synthetic); better
         * ignore
         */
        if (!_isIncludableField(mixinField)) {
            continue;
```

```
        }
        String name = mixinField.getName();
        // anything to mask? (if not, quietly ignore)
        AnnotatedField maskedField = fields.get(name);
        if (maskedField != null) {
            for (Annotation a : mixinField.getDeclaredAnnotations()) {
                if (_annotationIntrospector.isHandled(a)) {
                    maskedField.addOrOverride(a);
                }
            }
        }
    }
}

/*
/**********************************************************
/* Helper methods, constructing value types
/**********************************************************
 */

protected AnnotatedMethod _constructMethod(Method m)
{
    /* note: parameter annotations not used for regular (getter, setter)
     * methods; only for creator methods (static factory methods)
     */
    return new AnnotatedMethod(m, _collectRelevantAnnotations(m.getDeclaredAnnotations()), null);
}

protected AnnotatedConstructor _constructConstructor(Constructor<?> ctor, boolean defaultCtor)
{
    return new AnnotatedConstructor(ctor, _collectRelevantAnnotations(ctor.getDeclaredAnnotations()),
                                    defaultCtor ? null :  _collectRelevantAnnotations(ctor.getParameterAnnotations()));
}

protected AnnotatedMethod _constructCreatorMethod(Method m)
{
    return new AnnotatedMethod(m, _collectRelevantAnnotations(m.getDeclaredAnnotations()),
                               _collectRelevantAnnotations(m.getParameterAnnotations()));
}

protected AnnotatedField _constructField(Field f)
{
    return new AnnotatedField(f, _collectRelevantAnnotations(f.getDeclaredAnnotations()));
}

protected AnnotationMap[] _collectRelevantAnnotations(Annotation[][] anns)
{
    int len = anns.length;
```

```
    AnnotationMap[] result = new AnnotationMap[len];
    for (int i = 0; i < len; ++i) {
        result[i] = _collectRelevantAnnotations(anns[i]);
    }
    return result;
}

protected AnnotationMap _collectRelevantAnnotations(Annotation[] anns)
{
    AnnotationMap annMap = new AnnotationMap();
    if (anns != null) {
        for (Annotation a : anns) {
            if (_annotationIntrospector.isHandled(a)) {
                annMap.add(a);
            }
        }
    }
    return annMap;
}

/*
/**********************************************************
/* Helper methods, inclusion filtering
/**********************************************************
 */

protected boolean _isIncludableMethod(Method m, MethodFilter filter)
{
    if (filter != null && !filter.includeMethod(m)) {
        return false;
    }
    /* 07-Apr-2009, tatu: Looks like generics can introduce hidden
     *   bridge and/or synthetic methods. I don't think we want to
     *   consider those...
     */
    if (m.isSynthetic() || m.isBridge()) {
        return false;
    }
    return true;
}

private boolean _isIncludableField(Field f)
{
    /* I'm pretty sure synthetic fields are to be skipped...
     * (methods definitely are)
     */
    if (f.isSynthetic()) {
        return false;
```

```
    }
    // Static fields are never included, nor transient
    int mods = f.getModifiers();
    if (Modifier.isStatic(mods) || Modifier.isTransient(mods)) {
        return false;
    }
    return true;
}

/*
/**********************************************************
/* Helper methods, attaching annotations
/**********************************************************
 */

/**
 * @param addParamAnnotations Whether parameter annotations are to be
 *   added as well
 */
protected void _addMixOvers(Constructor<?> mixin, AnnotatedConstructor target,
        boolean addParamAnnotations)
{
    for (Annotation a : mixin.getDeclaredAnnotations()) {
        if (_annotationIntrospector.isHandled(a)) {
            target.addOrOverride(a);
        }
    }
    if (addParamAnnotations) {
        Annotation[][] pa = mixin.getParameterAnnotations();
        for (int i = 0, len = pa.length; i < len; ++i) {
            for (Annotation a : pa[i]) {
                target.addOrOverrideParam(i, a);
            }
        }
    }
}

/**
 * @param addParamAnnotations Whether parameter annotations are to be
 *   added as well
 */
protected void _addMixOvers(Method mixin, AnnotatedMethod target,
        boolean addParamAnnotations)
{
    for (Annotation a : mixin.getDeclaredAnnotations()) {
        if (_annotationIntrospector.isHandled(a)) {
            target.addOrOverride(a);
        }
```

```
        }
        if (addParamAnnotations) {
            Annotation[][] pa = mixin.getParameterAnnotations();
            for (int i = 0, len = pa.length; i < len; ++i) {
                for (Annotation a : pa[i]) {
                    target.addOrOverrideParam(i, a);
                }
            }
        }
    }

    /**
     * Method that will add annotations from specified source method to target method,
     * but only if target does not yet have them.
     */
    protected void _addMixUnders(Method src, AnnotatedMethod target)
    {
        for (Annotation a : src.getDeclaredAnnotations()) {
            if (_annotationIntrospector.isHandled(a)) {
                target.addIfNotPresent(a);
            }
        }
    }

    /*
    /**********************************************************
    /* Other methods
    /**********************************************************
     */

    @Override
    public String toString()
    {
        return "[AnnotedClass "+_class.getName()+"]";
    }
}
package org.codehaus.jackson.map.introspect;

import java.lang.reflect.Method;
import java.util.*;

/**
* Simple helper class used to keep track of collection of
* {@link AnnotatedMethod}s, accessible by lookup. Lookup
* is usually needed for augmenting and overriding annotations.
*/
public final class AnnotatedMethodMap
    implements Iterable<AnnotatedMethod>
```

```
{
  LinkedHashMap<MemberKey,AnnotatedMethod> _methods;

  public AnnotatedMethodMap() { }

  /**
   * Method called to add specified annotated method in the Map.
   */
  public void add(AnnotatedMethod am)
  {
     if (_methods == null) {
        _methods = new LinkedHashMap<MemberKey,AnnotatedMethod>();
     }
     _methods.put(new MemberKey(am.getAnnotated()), am);
  }

  /**
   * Method called to remove specified method, assuming
   * it exists in the Map
   */
  public AnnotatedMethod remove(AnnotatedMethod am)
  {
     return remove(am.getAnnotated());
  }

  public AnnotatedMethod remove(Method m)
  {
     if (_methods != null) {
        return _methods.remove(new MemberKey(m));
     }
     return null;
  }

  public boolean isEmpty() {
     return (_methods == null || _methods.size() == 0);
  }

  public int size() {
     return (_methods == null) ? 0 : _methods.size();
  }

  public AnnotatedMethod find(String name, Class<?>[] paramTypes)
  {
     if (_methods == null) {
        return null;
     }
     return _methods.get(new MemberKey(name, paramTypes));
  }
```

```
  public AnnotatedMethod find(Method m)
  {
     if (_methods == null) {
        return null;
     }
     return _methods.get(new MemberKey(m));
  }

  /*
  /**********************************************************
  /* Iterable implementation (for iterating over values)
  /**********************************************************
   */

  public Iterator<AnnotatedMethod> iterator()
  {
     if (_methods != null) {
        return _methods.values().iterator();
     }
     List<AnnotatedMethod> empty = Collections.emptyList();
     return empty.iterator();
  }
}
package org.codehaus.jackson.map.introspect;

import java.lang.reflect.*;

import org.codehaus.jackson.map.type.TypeBindings;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.type.JavaType;

public final class AnnotatedMethod
  extends AnnotatedWithParams
{
  final protected Method _method;

  // // Simple lazy-caching:

  protected Class<?>[] _paramTypes;

  /*
  /**********************************************************
  /* Life-cycle
  /**********************************************************
   */

  public AnnotatedMethod(Method method, AnnotationMap classAnn, AnnotationMap[] paramAnnotations)
```

```
{
   super(classAnn, paramAnnotations);
   _method = method;
}

/**
 * Method that constructs a new instance with settings (annotations, parameter annotations)
 * of this instance, but with different physical {@link Method}.
 *
 * @since 1.7
 */
public AnnotatedMethod withMethod(Method m)
{
   return new AnnotatedMethod(m, _annotations, _paramAnnotations);
}

/*
/******************************************************
/* Annotated impl
/******************************************************
 */

@Override
public Method getAnnotated() { return _method; }

@Override
public int getModifiers() { return _method.getModifiers(); }

@Override
public String getName() { return _method.getName(); }

/**
 * For methods, this returns declared return type, which is only
 * useful with getters (setters do not return anything; hence "void"
 * type is returned here)
 */
@Override
public Type getGenericType() {
   return _method.getGenericReturnType();
}

/**
 * For methods, this returns declared return type, which is only
 * useful with getters (setters do not return anything; hence "void"
 * type is returned here)
 */
@Override
public Class<?> getRawType() {
```

```
    return _method.getReturnType();
}

/**
 * As per [JACKSON-468], we need to also allow declaration of local
 * type bindings; mostly it will allow defining bounds.
 */
@Override
public JavaType getType(TypeBindings bindings)
{
    TypeVariable<?>[] localTypeParams = _method.getTypeParameters();
    // [JACKSON-468] Need to consider local type binding declarations too...
    if (localTypeParams != null && localTypeParams.length > 0) {
        bindings = bindings.childInstance();
        for (TypeVariable<?> var : localTypeParams) {
            String name = var.getName();
            // to prevent infinite loops, need to first add placeholder ("<T extends Enum<T>>" etc)
            bindings._addPlaceholder(name);
            // About only useful piece of information is the lower bound (which is at least Object.class)
            Type lowerBound = var.getBounds()[0];
            JavaType type = (lowerBound == null) ? TypeFactory.fastSimpleType(Object.class)
                    : TypeFactory.type(lowerBound, bindings);
            bindings.addBinding(var.getName(), type);
        }
    }
    return TypeFactory.type(getGenericType(), bindings);
}

/*
/**********************************************************
/* AnnotatedMember impl
/**********************************************************
 */

@Override
public Class<?> getDeclaringClass() { return _method.getDeclaringClass(); }

@Override
public Member getMember() { return _method; }

/*
/**********************************************************
/* Extended API, generic
/**********************************************************
 */

@Override
public AnnotatedParameter getParameter(int index) {
```

```
    return new AnnotatedParameter(this, getParameterType(index), _paramAnnotations[index]);
}

@Override
public int getParameterCount() {
    return getParameterTypes().length;
}

public Type[] getParameterTypes() {
    return _method.getGenericParameterTypes();
}

@Override
public Class<?> getParameterClass(int index)
{
    Class<?>[] types = _method.getParameterTypes();
    return (index >= types.length) ? null : types[index];
}

@Override
public Type getParameterType(int index)
{
    Type[] types = _method.getGenericParameterTypes();
    return (index >= types.length) ? null : types[index];
}

public Class<?>[] getParameterClasses()
{
    if (_paramTypes == null) {
        _paramTypes = _method.getParameterTypes();
    }
    return _paramTypes;
}

//public Type getGenericReturnType() { return _method.getGenericReturnType(); }

//public Class<?> getReturnType() { return _method.getReturnType(); }

public String getFullName() {
    return getDeclaringClass().getName() + "#" + getName() + "("
        +getParameterCount()+" params)";
}

/*
/**********************************************************
/* Extended API, specific annotations
/**********************************************************
 */
```

```
  @Override
  public String toString()
  {
     return "[method "+getName()+", annotations: "+_annotations+"]";
  }
}
package org.codehaus.jackson.map.introspect;

import java.lang.reflect.Method;

/**
* Simple interface that defines API used to filter out irrelevant
* methods
*/
public interface MethodFilter
{
  public boolean includeMethod(Method m);
}
package org.codehaus.jackson.map.introspect;

import org.codehaus.jackson.annotate.JsonAutoDetect;
import org.codehaus.jackson.annotate.JsonAutoDetect.Visibility;
import org.codehaus.jackson.annotate.JsonMethod;

import java.lang.reflect.Field;
import java.lang.reflect.Member;
import java.lang.reflect.Method;

/**
* Interface for object used for determine which property elements
* (methods, fields, constructors) can be auto-detected, with respect
* to their visibility modifiers.
*<p>
* Note on type declaration: funky recursive type is necessary to
* support builder/fluid pattern.
*
* @author tatu
* @since 1.5
*/
public interface VisibilityChecker<T extends VisibilityChecker<T>>
{
  // // Builder methods

  /**
   * Builder method that will return an instance that has same
   * settings as this instance has, except for values that
   * given annotation overrides.
```

```
 */
public T with(JsonAutoDetect ann);

/**
 * Builder method that will return a checker instance that has
 * specified minimum visibility level for regular ("getXxx") getters.
 */
public T withGetterVisibility(Visibility v);

/**
 * Builder method that will return a checker instance that has
 * specified minimum visibility level for "is-getters" ("isXxx").
 */
public T withIsGetterVisibility(Visibility v);

/**
 * Builder method that will return a checker instance that has
 * specified minimum visibility level for setters.
 */
public T withSetterVisibility(Visibility v);

/**
 * Builder method that will return a checker instance that has
 * specified minimum visibility level for creator methods
 * (constructors, factory methods)
 */
public T withCreatorVisibility(Visibility v);

/**
 * Builder method that will return a checker instance that has
 * specified minimum visibility level for fields.
 */
public T withFieldVisibility(Visibility v);

// // Accessors

/**
 * Method for checking whether given method is auto-detectable
 * as regular getter, with respect to its visibility (not considering
 * method signature or name, just visibility)
 */
public boolean isGetterVisible(Method m);
public boolean isGetterVisible(AnnotatedMethod m);

/**
 * Method for checking whether given method is auto-detectable
 * as is-getter, with respect to its visibility (not considering
 * method signature or name, just visibility)
```

```
 */
public boolean isIsGetterVisible(Method m);
public boolean isIsGetterVisible(AnnotatedMethod m);

/**
 * Method for checking whether given method is auto-detectable
 * as setter, with respect to its visibility (not considering
 * method signature or name, just visibility)
 */
public boolean isSetterVisible(Method m);
public boolean isSetterVisible(AnnotatedMethod m);

/**
 * Method for checking whether given method is auto-detectable
 * as Creator, with respect to its visibility (not considering
 * method signature or name, just visibility)
 */
public boolean isCreatorVisible(Member m);
public boolean isCreatorVisible(AnnotatedMember m);

/**
 * Method for checking whether given field is auto-detectable
 * as property, with respect to its visibility (not considering
 * method signature or name, just visibility)
 */
public boolean isFieldVisible(Field f);
public boolean isFieldVisible(AnnotatedField f);

/*
/**********************************************************
/* Standard implementation suitable for basic use
/**********************************************************
*/

/**
* Default standard implementation is purely based on visibility
* modifier of given class members, and its configured minimum
* levels.
* Implemented using "builder" (aka "Fluid") pattern, whereas instances
* are immutable, and configuration is achieved by chainable factory
* methods. As a result, type is declared is funky recursive generic
* type, to allow for sub-classing of build methods with property type
* co-variance.
*<p>
* Note on <code>JsonAutoDetect</code> annotation: it is used to
* access default minimum visibility access definitions.
*/
@JsonAutoDetect(
```

```
    getterVisibility = Visibility.PUBLIC_ONLY,
    isGetterVisibility = Visibility.PUBLIC_ONLY,
    setterVisibility = Visibility.ANY,
    /**
     * By default, all matching single-arg constructed are found,
     * regardless of visibility. Does not apply to factory methods,
     * they can not be auto-detected; ditto for multiple-argument
     * constructors.
     */
    creatorVisibility = Visibility.ANY,
    fieldVisibility = Visibility.PUBLIC_ONLY
  )
  public static class Std
    implements VisibilityChecker<Std>
  {
    /**
     * This is the canonical base instance, configured with default
     * visibility values
     */
    protected final static Std DEFAULT = new Std(Std.class.getAnnotation(JsonAutoDetect.class));

    protected final Visibility _getterMinLevel;
    protected final Visibility _isGetterMinLevel;
    protected final Visibility _setterMinLevel;
    protected final Visibility _creatorMinLevel;
    protected final Visibility _fieldMinLevel;

    public static Std defaultInstance() { return DEFAULT; }

    /**
     * Constructor used for building instance that has minumum visibility
     * levels as indicated by given annotation instance
     *
     * @param ann Annotations to use for determining minimum visibility levels
     */
  public Std(JsonAutoDetect ann)
  {
    JsonMethod[] incl = ann.value();
    // let's combine checks for enabled/disabled, with minimimum level checks:
    _getterMinLevel = hasMethod(incl, JsonMethod.GETTER) ? ann.getterVisibility() : Visibility.NONE;
        _isGetterMinLevel = hasMethod(incl, JsonMethod.IS_GETTER) ? ann.isGetterVisibility() :
Visibility.NONE;
        _setterMinLevel = hasMethod(incl, JsonMethod.SETTER) ? ann.setterVisibility() : Visibility.NONE;
        _creatorMinLevel = hasMethod(incl, JsonMethod.CREATOR) ? ann.creatorVisibility() : Visibility.NONE;
        _fieldMinLevel = hasMethod(incl, JsonMethod.FIELD) ? ann.fieldVisibility() : Visibility.NONE;
  }

  /**
```

```
 * Constructor that allows directly specifying minimum visibility levels to use
 */
public Std(Visibility getter, Visibility isGetter, Visibility setter, Visibility creator, Visibility field)
     {
   _getterMinLevel = getter;
   _isGetterMinLevel = isGetter;
   _setterMinLevel = setter;
   _creatorMinLevel = creator;
   _fieldMinLevel = field;
     }

     /*
     /**********************************************************
/* Builder/fluid methods for instantiating configured
/* instances
/**********************************************************
 */

public Std with(JsonAutoDetect ann)
{
   if (ann == null) return this;
   Std curr = this;

   JsonMethod[] incl = ann.value();
   Visibility v;

   v = hasMethod(incl, JsonMethod.GETTER) ? ann.getterVisibility() : Visibility.NONE;
   curr = curr.withGetterVisibility(v);
   v = hasMethod(incl, JsonMethod.IS_GETTER) ? ann.isGetterVisibility() : Visibility.NONE;
   curr = curr.withIsGetterVisibility(v);
   v = hasMethod(incl, JsonMethod.SETTER) ? ann.setterVisibility() : Visibility.NONE;
        curr  = curr.withSetterVisibility(v);
        v = hasMethod(incl, JsonMethod.CREATOR) ? ann.creatorVisibility() : Visibility.NONE;
        curr = curr.withCreatorVisibility(v);
        v = hasMethod(incl, JsonMethod.FIELD) ? ann.fieldVisibility() : Visibility.NONE;
        curr = curr.withFieldVisibility(v);
   return curr;
}

public Std withGetterVisibility(Visibility v) {
   if (v == Visibility.DEFAULT)  v = DEFAULT._getterMinLevel;
        if (_getterMinLevel == v) return this;
   return new Std(v, _isGetterMinLevel, _setterMinLevel, _creatorMinLevel, _fieldMinLevel);
}

     public Std withIsGetterVisibility(Visibility v) {
        if (v == Visibility.DEFAULT)  v = DEFAULT._isGetterMinLevel;
        if (_isGetterMinLevel == v) return this;
```

```
        return new Std(_getterMinLevel, v, _setterMinLevel, _creatorMinLevel, _fieldMinLevel);
    }

public Std withSetterVisibility(Visibility v) {
        if (v == Visibility.DEFAULT)  v = DEFAULT._setterMinLevel;
        if (_setterMinLevel == v) return this;
        return new Std(_getterMinLevel, _isGetterMinLevel, v, _creatorMinLevel, _fieldMinLevel);
}

public Std withCreatorVisibility(Visibility v) {
        if (v == Visibility.DEFAULT)  v = DEFAULT._creatorMinLevel;
        if (_creatorMinLevel == v) return this;
   return new Std(_getterMinLevel, _isGetterMinLevel, _setterMinLevel, v, _fieldMinLevel);
}

public Std withFieldVisibility(Visibility v) {
        if (v == Visibility.DEFAULT)  v = DEFAULT._fieldMinLevel;
        if (_fieldMinLevel == v) return this;
   return new Std(_getterMinLevel, _isGetterMinLevel, _setterMinLevel, _creatorMinLevel, v);
}

    /*
    /**********************************************************
    /* Public API impl
    /**********************************************************
     */

public boolean isCreatorVisible(Member m) {
   return _creatorMinLevel.isVisible(m);
  }

  public boolean isCreatorVisible(AnnotatedMember m) {
     return isCreatorVisible(m.getMember());
  }

  public boolean isFieldVisible(Field f) {
     return _fieldMinLevel.isVisible(f);
  }

  public boolean isFieldVisible(AnnotatedField f) {
     return isFieldVisible(f.getAnnotated());
  }

  public boolean isGetterVisible(Method m) {
     return _getterMinLevel.isVisible(m);
  }
  public boolean isGetterVisible(AnnotatedMethod m) {
     return isGetterVisible(m.getAnnotated());
```

```
    }

    public boolean isIsGetterVisible(Method m) {
        return _isGetterMinLevel.isVisible(m);
    }

    public boolean isIsGetterVisible(AnnotatedMethod m) {
        return isIsGetterVisible(m.getAnnotated());
    }

    public boolean isSetterVisible(Method m) {
        return _setterMinLevel.isVisible(m);
    }

    public boolean isSetterVisible(AnnotatedMethod m) {
        return isSetterVisible(m.getAnnotated());
    }

    /*
    /**********************************************************
    /* Helper methods
    /**********************************************************
     */

    private static boolean hasMethod(JsonMethod[] methods, JsonMethod method)
    {
        for (JsonMethod curr : methods) {
            if (curr == method || curr == JsonMethod.ALL) return true;
        }
        return false;
    }

    /*
    /**********************************************************
    /* Standard methods
    /**********************************************************
     */
    @Override
    public String toString() {
        return new StringBuilder("[Visibility:")
            .append(" getter: ").append(_getterMinLevel)
            .append(", isGetter: ").append(_isGetterMinLevel)
            .append(", setter: ").append(_setterMinLevel)
            .append(", creator: ").append(_creatorMinLevel)
            .append(", field: ").append(_fieldMinLevel)
            .append("]").toString();
    }
}
```

```
}
package org.codehaus.jackson.map.introspect;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;

/**
* Helper class needed to be able to efficiently access class
* member functions ({@link Method}s and {@link Constructor}s)
* in {@link java.util.Map}s.
*/
public final class MemberKey
{
  final static Class<?>[] NO_CLASSES = new Class<?>[0];

  final String _name;
  final Class<?>[] _argTypes;

  public MemberKey(Method m)
  {
    this(m.getName(), m.getParameterTypes());
  }

  public MemberKey(Constructor<?> ctor)
  {
    this("", ctor.getParameterTypes());
  }

  public MemberKey(String name, Class<?>[] argTypes)
  {
    _name = name;
    _argTypes = (argTypes == null) ? NO_CLASSES : argTypes;
  }

  @Override
  public String toString() {
    return _name + "(" + _argTypes.length+"-args)";
  }

  @Override
  public int hashCode()
  {
    return _name.hashCode() + _argTypes.length;
  }

  @Override
  public boolean equals(Object o)
  {
```

```
        if (o == this) return true;
        if (o == null) return false;
        if (o.getClass() != getClass()) {
            return false;
        }
        MemberKey other = (MemberKey) o;
        if (!_name.equals(other._name)) {
            return false;
        }
        Class<?>[] otherArgs = other._argTypes;
        int len = _argTypes.length;
        if (otherArgs.length != len) {
            return false;
        }
        for (int i = 0; i < len; ++i) {
            Class<?> type1 = otherArgs[i];
            Class<?> type2 = _argTypes[i];
            if (type1 == type2) {
                continue;
            }
            /* 23-Feb-2009, tatu: Are there any cases where we would have to
             *   consider some narrowing conversions or such? For now let's
             *   assume exact type match is enough
             */
            /* 07-Apr-2009, tatu: Indeed there are (see [JACKSON-97]).
             *    This happens with generics when a bound is specified.
             *    I hope this works; check here must be transitive
             */
            if (type1.isAssignableFrom(type2) || type2.isAssignableFrom(type1)) {
                continue;
            }
            return false;
        }
        return true;
    }
}
package org.codehaus.jackson.map.introspect;

import java.lang.annotation.Annotation;
import java.util.ArrayList;
import java.util.List;

import org.codehaus.jackson.annotate.*;
import org.codehaus.jackson.map.AnnotationIntrospector;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.KeyDeserializer;
```

```
import org.codehaus.jackson.map.annotate.JsonCachable;
import org.codehaus.jackson.map.annotate.JsonDeserialize;
import org.codehaus.jackson.map.annotate.JsonFilter;
import org.codehaus.jackson.map.annotate.JsonSerialize;
import org.codehaus.jackson.map.annotate.JsonTypeIdResolver;
import org.codehaus.jackson.map.annotate.JsonTypeResolver;
import org.codehaus.jackson.map.annotate.JsonView;
import org.codehaus.jackson.map.annotate.NoClass;
import org.codehaus.jackson.map.jsontype.NamedType;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.map.jsontype.impl.StdTypeResolverBuilder;
import org.codehaus.jackson.map.ser.impl.RawSerializer;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.type.JavaType;

/**
* {@link AnnotationIntrospector} implementation that handles standard
* Jackson annotations.
*/
public class JacksonAnnotationIntrospector
  extends AnnotationIntrospector
{
  public JacksonAnnotationIntrospector() { }

  /*
  /**********************************************************
  /* General annotation properties
  /**********************************************************
   */

  @Override
  public boolean isHandled(Annotation ann)
  {
    Class<? extends Annotation> acls = ann.annotationType();

    /* 16-May-2009, tatu: used to check this like so...
      final String JACKSON_PKG_PREFIX = "org.codehaus.jackson";

      Package pkg = acls.getPackage();
      return (pkg != null) && (pkg.getName().startsWith(JACKSON_PKG_PREFIX));
    */

    // but this is more reliable, now that we have tag annotation:
    return acls.getAnnotation(JacksonAnnotation.class) != null;
  }

  /*
```

```
/**********************************************************
/* General annotations
/**********************************************************
 */

@Override
public String findEnumValue(Enum<?> value)
{
   return value.name();
}

/*
/**********************************************************
/* General class annotations
/**********************************************************
 */

@Override
public Boolean findCachability(AnnotatedClass ac)
{
   JsonCachable ann = ac.getAnnotation(JsonCachable.class);
   if (ann == null) {
      return null;
   }
   return ann.value() ? Boolean.TRUE : Boolean.FALSE;
}

@Override
public String findRootName(AnnotatedClass ac)
{
   // No annotation currently defined for this feature, so:
   return null;
}

@Override
public String[] findPropertiesToIgnore(AnnotatedClass ac) {
   JsonIgnoreProperties ignore = ac.getAnnotation(JsonIgnoreProperties.class);
   return (ignore == null) ? null : ignore.value();
}

@Override
public Boolean findIgnoreUnknownProperties(AnnotatedClass ac) {
   JsonIgnoreProperties ignore = ac.getAnnotation(JsonIgnoreProperties.class);
   return (ignore == null) ? null : ignore.ignoreUnknown();
}

@Override
public Boolean isIgnorableType(AnnotatedClass ac) {
```

```
    JsonIgnoreType ignore = ac.getAnnotation(JsonIgnoreType.class);
    return (ignore == null) ? null : ignore.value();
}

@Override
public Object findFilterId(AnnotatedClass ac)
{
    JsonFilter ann = ac.getAnnotation(JsonFilter.class);
    if (ann != null) {
        String id = ann.value();
        // Empty String is same as not having annotation, to allow overrides
        if (id.length() > 0) {
            return id;
        }
    }
    return null;
}

/*
/**********************************************************
/* Property auto-detection
/**********************************************************
 */

@Override
public VisibilityChecker<?> findAutoDetectVisibility(AnnotatedClass ac,
    VisibilityChecker<?> checker)
{
    JsonAutoDetect ann = ac.getAnnotation(JsonAutoDetect.class);
    return (ann == null) ? checker : checker.with(ann);
}

/*
/**********************************************************
/* General member (field, method/constructor) annotations
/**********************************************************
 */

// @since 1.6
@Override
public ReferenceProperty findReferenceType(AnnotatedMember member)
{
    JsonManagedReference ref1 = member.getAnnotation(JsonManagedReference.class);
    if (ref1 != null) {
        return AnnotationIntrospector.ReferenceProperty.managed(ref1.value());
    }
    JsonBackReference ref2 = member.getAnnotation(JsonBackReference.class);
    if (ref2 != null) {
```

```
            return AnnotationIntrospector.ReferenceProperty.back(ref2.value());
        }
        return null;
    }

    /*
    /**********************************************************
    /* Class annotations for PM type handling (1.5+)
    /**********************************************************
     */

    @Override
    public TypeResolverBuilder<?> findTypeResolver(AnnotatedClass ac, JavaType baseType)
    {
        return _findTypeResolver(ac, baseType);
    }

    /**
     * Since 1.7, it is possible to use {@link JsonTypeInfo} from a property too.
     */
    @Override
    public TypeResolverBuilder<?> findPropertyTypeResolver(AnnotatedMember am, JavaType baseType)
    {
        /* As per definition of @JsonTypeInfo, should only apply to contents of container
         * (collection, map) types, not container types themselves:
         */
        if (baseType.isContainerType()) return null;
        // No per-member type overrides (yet)
        return _findTypeResolver(am, baseType);
    }

    /**
     * Since 1.7, it is possible to use {@link JsonTypeInfo} from a property too.
     */
    @Override
    public TypeResolverBuilder<?> findPropertyContentTypeResolver(AnnotatedMember am, JavaType
containerType)
    {
        /* First: let's ensure property is a container type: caller should have
         * verified but just to be sure
         */
        if (!containerType.isContainerType()) {
            throw new IllegalArgumentException("Must call method with a container type (got "+containerType+")");
        }
        return _findTypeResolver(am, containerType);
    }

    @Override
```

```
public List<NamedType> findSubtypes(Annotated a)
{
   JsonSubTypes t = a.getAnnotation(JsonSubTypes.class);
   if (t == null) return null;
   JsonSubTypes.Type[] types = t.value();
   ArrayList<NamedType> result = new ArrayList<NamedType>(types.length);
   for (JsonSubTypes.Type type : types) {
      result.add(new NamedType(type.value(), type.name()));
   }
   return result;
}

@Override
public String findTypeName(AnnotatedClass ac)
{
   JsonTypeName tn = ac.getAnnotation(JsonTypeName.class);
   return (tn == null) ? null : tn.value();
}

/*
/**********************************************************
/* General method annotations
/**********************************************************
*/

@Override
public boolean isIgnorableMethod(AnnotatedMethod m) {
   return _isIgnorable(m);
}

@Override
public boolean isIgnorableConstructor(AnnotatedConstructor c) {
   return _isIgnorable(c);
}

/*
/**********************************************************
/* General field annotations
/**********************************************************
 */

@Override
public boolean isIgnorableField(AnnotatedField f) {
   return _isIgnorable(f);
}

/*
/**********************************************************
```

```
/* Serialization: general annotations
/**********************************************************
*/

@Override
public Object findSerializer(Annotated a, BeanProperty property)
{
   /* 21-May-2009, tatu: Slight change; primary annotation is now
    *    @JsonSerialize; @JsonUseSerializer is deprecated
    */
   JsonSerialize ann = a.getAnnotation(JsonSerialize.class);
   if (ann != null) {
      Class<? extends JsonSerializer<?>> serClass = ann.using();
      if (serClass != JsonSerializer.None.class) {
         return serClass;
      }
   }

   /* 18=Oct-2010, tatu: [JACKSON-351] @JsonRawValue handled just here, for now;
    *  if we need to get raw indicator from other sources need to add
    *  separate accessor within {@link AnnotationIntrospector} interface.
    */
   JsonRawValue annRaw =  a.getAnnotation(JsonRawValue.class);
   if ((annRaw != null) && annRaw.value()) {
      // let's construct instance with nominal type:
      Class<?> cls = a.getRawType();
      return new RawSerializer<Object>(cls);
   }
   return null;
}

@SuppressWarnings("deprecation")
@Override
public JsonSerialize.Inclusion findSerializationInclusion(Annotated a, JsonSerialize.Inclusion defValue)
{
   JsonSerialize ann = a.getAnnotation(JsonSerialize.class);
   if (ann != null) {
      return ann.include();
   }
   /* 23-May-2009, tatu: Will still support now-deprecated (as of 1.1)
    *   legacy annotation too:
    */
   JsonWriteNullProperties oldAnn = a.getAnnotation(JsonWriteNullProperties.class);
   if (oldAnn != null) {
      boolean writeNulls = oldAnn.value();
      return writeNulls ? JsonSerialize.Inclusion.ALWAYS : JsonSerialize.Inclusion.NON_NULL;
   }
   return defValue;
```

```
}

@Override
public Class<?> findSerializationType(Annotated am)
{
   // Primary annotation, JsonSerialize
   JsonSerialize ann = am.getAnnotation(JsonSerialize.class);
   if (ann != null) {
      Class<?> cls = ann.as();
      if (cls != NoClass.class) {
         return cls;
      }
   }
   return null;
}

@Override
public JsonSerialize.Typing findSerializationTyping(Annotated a)
{
   JsonSerialize ann = a.getAnnotation(JsonSerialize.class);
   return (ann == null) ? null : ann.typing();
}

@Override
public Class<?>[] findSerializationViews(Annotated a)
{
   JsonView ann = a.getAnnotation(JsonView.class);
   return (ann == null) ? null : ann.value();
}

/*
/**********************************************************
/* Serialization: class annotations
/**********************************************************
 */

@Override
public String[] findSerializationPropertyOrder(AnnotatedClass ac) {
   JsonPropertyOrder order = ac.getAnnotation(JsonPropertyOrder.class);
   return (order == null) ? null : order.value();
}

@Override
public Boolean findSerializationSortAlphabetically(AnnotatedClass ac) {
   JsonPropertyOrder order = ac.getAnnotation(JsonPropertyOrder.class);
   return (order == null) ? null : order.alphabetic();
}
```

```
/*
/**********************************************************
/* Serialization: method annotations
/**********************************************************
 */

@SuppressWarnings("deprecation")
@Override
public String findGettablePropertyName(AnnotatedMethod am)
{
   /* 22-May-2009, tatu: JsonProperty is the primary annotation
    *   to check for
    */
   JsonProperty pann = am.getAnnotation(JsonProperty.class);
   if (pann != null) {
      return pann.value();
   }
   /* 22-May-2009, tatu: JsonGetter is deprecated as of 1.1
    *    but still supported
    */
   JsonGetter ann = am.getAnnotation(JsonGetter.class);
   if (ann != null) {
      return ann.value();
   }
   /* 22-May-2009, tatu: And finally, JsonSerialize implies
    *   that there is a property, although doesn't define name
    */
   // 09-Apr-2010, tatu: Ditto for JsonView
   if (am.hasAnnotation(JsonSerialize.class) || am.hasAnnotation(JsonView.class)) {
      return "";
   }
   return null;
}

@Override
public boolean hasAsValueAnnotation(AnnotatedMethod am)
{
   JsonValue ann = am.getAnnotation(JsonValue.class);
   // value of 'false' means disabled...
   return (ann != null && ann.value());
}

/*
/**********************************************************
/* Serialization: field annotations
/**********************************************************
*/
```

```
@Override
public String findSerializablePropertyName(AnnotatedField af)
{
   JsonProperty pann = af.getAnnotation(JsonProperty.class);
   if (pann != null) {
      return pann.value();
   }
   // Also: having JsonSerialize implies it is such a property
   // 09-Apr-2010, tatu: Ditto for JsonView
   if (af.hasAnnotation(JsonSerialize.class) || af.hasAnnotation(JsonView.class)) {
      return "";
   }
   return null;
}

/*
/**********************************************************
/* Deserialization: general annotations
/**********************************************************
 */

@Override
public Class<? extends JsonDeserializer<?>> findDeserializer(Annotated a, BeanProperty property)
{
   /* 21-May-2009, tatu: Slight change; primary annotation is now
    *    @JsonDeserialize; @JsonUseDeserializer is deprecated
    */
   JsonDeserialize ann = a.getAnnotation(JsonDeserialize.class);
   if (ann != null) {
      Class<? extends JsonDeserializer<?>> deserClass = ann.using();
      if (deserClass != JsonDeserializer.None.class) {
         return deserClass;
      }
   }
   // 31-Jan-2010, tatus: @JsonUseDeserializer removed as of 1.5
   return null;
}

@Override
public Class<? extends KeyDeserializer> findKeyDeserializer(Annotated a)
{
   JsonDeserialize ann = a.getAnnotation(JsonDeserialize.class);
   if (ann != null) {
      Class<? extends KeyDeserializer> deserClass = ann.keyUsing();
      if (deserClass != KeyDeserializer.None.class) {
         return deserClass;
      }
   }
```

```
    return null;
}

@Override
public Class<? extends JsonDeserializer<?>> findContentDeserializer(Annotated a)
{
    JsonDeserialize ann = a.getAnnotation(JsonDeserialize.class);
    if (ann != null) {
        Class<? extends JsonDeserializer<?>> deserClass = ann.contentUsing();
        if (deserClass != JsonDeserializer.None.class) {
            return deserClass;
        }
    }
    return null;
}

@Override
public Class<?> findDeserializationType(Annotated am, JavaType baseType,
        String propName)
{
    // Primary annotation, JsonDeserialize
    JsonDeserialize ann = am.getAnnotation(JsonDeserialize.class);
    if (ann != null) {
        Class<?> cls = ann.as();
        if (cls != NoClass.class) {
            return cls;
        }
    }

    /* TODO: !!! 21-May-2009, tatu: JsonClass is deprecated; will need to
     *    drop support at a later point (for 2.0?)
     */
    @SuppressWarnings("deprecation")
    JsonClass oldAnn = am.getAnnotation(JsonClass.class);
    if (oldAnn != null) {
        @SuppressWarnings("deprecation")
        Class<?> cls = oldAnn.value();
        if(cls != NoClass.class) {
            return cls;
        }
    }
    return null;
}

@Override
public Class<?> findDeserializationKeyType(Annotated am, JavaType baseKeyType,
        String propName)
{
```

```
    // Primary annotation, JsonDeserialize
    JsonDeserialize ann = am.getAnnotation(JsonDeserialize.class);
    if (ann != null) {
        Class<?> cls = ann.keyAs();
        if (cls != NoClass.class) {
            return cls;
        }
    }

    /* !!! 21-May-2009, tatu: JsonClass is deprecated; will need to
     *    drop support at a later point (for 2.0?)
     */
    @SuppressWarnings("deprecation")
    JsonKeyClass oldAnn = am.getAnnotation(JsonKeyClass.class);
    if (oldAnn != null) {
        @SuppressWarnings("deprecation")
        Class<?> cls = oldAnn.value();
        if(cls != NoClass.class) {
            return cls;
        }
    }
    return null;
}

@SuppressWarnings("deprecation")
@Override
public Class<?> findDeserializationContentType(Annotated am, JavaType baseContentType,
        String propName)
{
    // Primary annotation, JsonDeserialize
    JsonDeserialize ann = am.getAnnotation(JsonDeserialize.class);
    if (ann != null) {
        Class<?> cls = ann.contentAs();
        if (cls != NoClass.class) {
            return cls;
        }
    }

    /* !!! 21-May-2009, tatu: JsonClass is deprecated; will need to
     *    drop support at a later point (for 2.0?)
     */
    JsonContentClass oldAnn = am.getAnnotation(JsonContentClass.class);
    if (oldAnn != null) {
        Class<?> cls = oldAnn.value();
        if(cls != NoClass.class) {
            return cls;
        }
    }
```

```
    return null;
}

/*
/**********************************************************
/* Deserialization: Method annotations
/**********************************************************
 */

@Override
public String findSettablePropertyName(AnnotatedMethod am)
{
    /* 16-Apr-2010, tatu: Existing priority (since 1.1) is that
     *   @JsonProperty is checked first; and @JsonSetter next.
     *   This is not quite optimal now that @JsonSetter is un-deprecated.
     *   However, it is better to have stable behavior rather than
     *   cause compatibility problems by fine-tuning.
     */
    JsonProperty pann = am.getAnnotation(JsonProperty.class);
    if (pann != null) {
        return pann.value();
    }
    JsonSetter ann = am.getAnnotation(JsonSetter.class);
    if (ann != null) {
        return ann.value();
    }
    /* 22-May-2009, tatu: And finally, JsonSerialize implies
     *   that there is a property, although doesn't define name
     */
    // 09-Apr-2010, tatu: Ditto for JsonView
 if (am.hasAnnotation(JsonDeserialize.class) || am.hasAnnotation(JsonView.class)) {
        return "";
    }
    return null;
}

@Override
public boolean hasAnySetterAnnotation(AnnotatedMethod am)
{
    /* No dedicated disabling; regular @JsonIgnore used
     * if needs to be ignored (and if so, is handled prior
     * to this method getting called)
     */
    return am.hasAnnotation(JsonAnySetter.class);
}

@Override
public boolean hasAnyGetterAnnotation(AnnotatedMethod am)
```

```
{
   /* No dedicated disabling; regular @JsonIgnore used
    * if needs to be ignored (handled separately
    */
   return am.hasAnnotation(JsonAnyGetter.class);
}

@Override
public boolean hasCreatorAnnotation(Annotated a)
{
   /* No dedicated disabling; regular @JsonIgnore used
    * if needs to be ignored (and if so, is handled prior
    * to this method getting called)
    */
   return a.hasAnnotation(JsonCreator.class);
}

/*
/**********************************************************
/* Deserialization: field annotations
/**********************************************************
 */

@Override
public String findDeserializablePropertyName(AnnotatedField af)
{
   JsonProperty pann = af.getAnnotation(JsonProperty.class);
   if (pann != null) {
      return pann.value();
   }
   // Also: having JsonDeserialize implies it is such a property
   // 09-Apr-2010, tatu: Ditto for JsonView
   if (af.hasAnnotation(JsonDeserialize.class) || af.hasAnnotation(JsonView.class)) {
      return "";
   }
   return null;
}

/*
/**********************************************************
/* Deserialization: parameters annotations
/**********************************************************
 */

@Override
public String findPropertyNameForParam(AnnotatedParameter param)
{
   if (param != null) {
```

```
        JsonProperty pann = param.getAnnotation(JsonProperty.class);
        if (pann != null) {
            return pann.value();
        }
        /* And can not use JsonDeserialize as we can not use
         * name auto-detection (names of local variables including
         * parameters are not necessarily preserved in bytecode)
         */
    }
    return null;
}

/*
/**********************************************************
/* Helper methods
/**********************************************************
 */

protected boolean _isIgnorable(Annotated a)
{
    JsonIgnore ann = a.getAnnotation(JsonIgnore.class);
    return (ann != null && ann.value());
}

/**
 * Helper method called to construct and initialize instance of {@link TypeResolverBuilder}
 * if given annotated element indicates one is needed.
 */
protected TypeResolverBuilder<?> _findTypeResolver(Annotated ann, JavaType baseType)
{
    // First: maybe we have explicit type resolver?
    TypeResolverBuilder<?> b;
    JsonTypeInfo info = ann.getAnnotation(JsonTypeInfo.class);
    JsonTypeResolver resAnn = ann.getAnnotation(JsonTypeResolver.class);
    if (resAnn != null) {
        /* 14-Aug-2010, tatu: not sure if this can ever happen normally, but unit
         *    tests were able to trigger this... so let's check:
         */
        if (info == null) {
            return null;
        }
        /* let's not try to force access override (would need to pass
         * settings through if we did, since that's not doable on some
         * platforms)
         */
        b = ClassUtil.createInstance(resAnn.value(), false);
    } else { // if not, use standard one, if indicated by annotations
        if (info == null || info.use() == JsonTypeInfo.Id.NONE) {
```

```
            return null;
        }
        b = _constructStdTypeResolverBuilder();
    }
    // Does it define a custom type id resolver?
    JsonTypeIdResolver idResInfo = ann.getAnnotation(JsonTypeIdResolver.class);
    TypeIdResolver idRes = (idResInfo == null) ? null
            : ClassUtil.createInstance(idResInfo.value(), true);
    if (idRes != null) { // [JACKSON-359]
        idRes.init(baseType);
    }
    b = b.init(info.use(), idRes);
    b = b.inclusion(info.include());
    b = b.typeProperty(info.property());
    return b;
  }

  /**
   * Helper method for constructing standard {@link TypeResolverBuilder}
   * implementation.
   *
   * @since 1.7
   */
  protected StdTypeResolverBuilder _constructStdTypeResolverBuilder()
  {
    return new StdTypeResolverBuilder();
  }

}
package org.codehaus.jackson.map.introspect;

import java.lang.annotation.Annotation;
import java.util.*;

import org.codehaus.jackson.map.util.Annotations;

/**
* Simple helper class used to keep track of collection of
* Jackson Annotations associated with annotatable things
* (methods, constructors, classes).
* Note that only Jackson-owned annotations are tracked (for now?).
*/
public final class AnnotationMap implements Annotations
{
  protected HashMap<Class<? extends Annotation>,Annotation> _annotations;

  public AnnotationMap() { }
```

```
@SuppressWarnings("unchecked")
public <A extends Annotation> A get(Class<A> cls)
{
   if (_annotations == null) {
      return null;
   }
   return (A) _annotations.get(cls);
}

public int size() {
   return (_annotations == null) ? 0 : _annotations.size();
}

/**
 * Method called to add specified annotation in the Map, but
 * only if it didn't yet exist.
 */
public void addIfNotPresent(Annotation ann)
{
   if (_annotations == null || !_annotations.containsKey(ann.annotationType())) {
      _add(ann);
   }
}

/**
 * Method called to add specified annotation in the Map.
 */
public void add(Annotation ann) {
   _add(ann);
}

@Override
public String toString()
{
   if (_annotations == null) {
      return "[null]";
   }
   return _annotations.toString();
}

/*
/**********************************************************
/* Helper methods
/**********************************************************
 */

protected final void _add(Annotation ann)
{
```

```
        if (_annotations == null) {
            _annotations = new HashMap<Class<? extends Annotation>,Annotation>();
        }
        _annotations.put(ann.annotationType(), ann);
    }
}
package org.codehaus.jackson.map.introspect;

import java.lang.annotation.Annotation;
import java.lang.reflect.Type;

/**
* Intermediate base class that encapsulates features that
* constructors and methods share.
*/
public abstract class AnnotatedWithParams
    extends AnnotatedMember
{
    /**
     * Annotations directly associated with the annotated
     * entity.
     */
    protected final AnnotationMap _annotations;

    /**
     * Annotations associated with parameters of the annotated
     * entity (method or constructor parameters)
     */
    protected final AnnotationMap[] _paramAnnotations;

    /*
    /**********************************************************
    /* Life-cycle
    /**********************************************************
     */

    protected AnnotatedWithParams(AnnotationMap classAnn, AnnotationMap[] paramAnnotations)
    {
        _annotations = classAnn;
        _paramAnnotations = paramAnnotations;
    }

    /**
     * Method called to override a class annotation, usually due to a mix-in
     * annotation masking or overriding an annotation 'real' class
     */
    public final void addOrOverride(Annotation a)
    {
```

```
    _annotations.add(a);
}

/**
 * Method called to override a method parameter annotation,
 * usually due to a mix-in
 * annotation masking or overriding an annotation 'real' method
 * has.
 */
public final void addOrOverrideParam(int paramIndex, Annotation a)
{
    AnnotationMap old = _paramAnnotations[paramIndex];
    if (old == null) {
        old = new AnnotationMap();
        _paramAnnotations[paramIndex] = old;
    }
    old.add(a);
}

/**
 * Method called to augment annotations, by adding specified
 * annotation if and only if it is not yet present in the
 * annotation map we have.
 */
public final void addIfNotPresent(Annotation a)
{
    _annotations.addIfNotPresent(a);
}

/*
/**********************************************************
/* Partial Annotated impl
/**********************************************************
 */

@Override
public final <A extends Annotation> A getAnnotation(Class<A> acls)
{
    return _annotations.get(acls);
}

/*
/**********************************************************
/* Extended API
/**********************************************************
 */

public final AnnotationMap getParameterAnnotations(int index)
```

```
  {
    if (_paramAnnotations != null) {
      if (index >= 0 && index <= _paramAnnotations.length) {
        return _paramAnnotations[index];
      }
    }
    return null;
  }

  public abstract AnnotatedParameter getParameter(int index);

  public abstract int getParameterCount();

  public abstract Class<?> getParameterClass(int index);

  public abstract Type getParameterType(int index);

  public final int getAnnotationCount() { return _annotations.size(); }
}
package org.codehaus.jackson.map.introspect;

import java.lang.reflect.Constructor;
import java.lang.reflect.Member;
import java.lang.reflect.Type;
import java.lang.reflect.TypeVariable;

import org.codehaus.jackson.map.type.TypeBindings;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.type.JavaType;

public final class AnnotatedConstructor
  extends AnnotatedWithParams
{
  protected final Constructor<?> _constructor;

  /*
  /**********************************************************
  /* Life-cycle
  /**********************************************************
   */

  public AnnotatedConstructor(Constructor<?> constructor,
                              AnnotationMap classAnn, AnnotationMap[] paramAnn)
  {
    super(classAnn, paramAnn);
    if (constructor == null) {
      throw new IllegalArgumentException("Null constructor not allowed");
    }
```

```
    _constructor = constructor;
}

/*
/**********************************************************
/* Annotated impl
/**********************************************************
 */

@Override
public Constructor<?> getAnnotated() { return _constructor; }

@Override
public int getModifiers() { return _constructor.getModifiers(); }

@Override
public String getName() { return _constructor.getName(); }

@Override
public Type getGenericType() {
    return getRawType();
}

@Override
public Class<?> getRawType() {
    return _constructor.getDeclaringClass();
}

// note: copied verbatim from AnnotatedMethod; hard to generalize
/**
 * As per [JACKSON-468], we need to also allow declaration of local
 * type bindings; mostly it will allow defining bounds.
 */
@Override
public JavaType getType(TypeBindings bindings)
{
    TypeVariable<?>[] localTypeParams = _constructor.getTypeParameters();
    // [JACKSON-468] Need to consider local type binding declarations too...
    if (localTypeParams != null && localTypeParams.length > 0) {
        bindings = bindings.childInstance();
        for (TypeVariable<?> var : localTypeParams) {
            String name = var.getName();
            // to prevent infinite loops, need to first add placeholder ("<T extends Enum<T>>" etc)
            bindings._addPlaceholder(name);
            // About only useful piece of information is the lower bound (which is at least Object.class)
            Type lowerBound = var.getBounds()[0];
            JavaType type = (lowerBound == null) ? TypeFactory.fastSimpleType(Object.class)
                    : TypeFactory.type(lowerBound, bindings);
```

```
            bindings.addBinding(var.getName(), type);
        }
    }
    return TypeFactory.type(getGenericType(), bindings);
}

/*
/**********************************************************
/* Extended API
/**********************************************************
 */

@Override
public AnnotatedParameter getParameter(int index) {
    return new AnnotatedParameter(this, getParameterType(index), _paramAnnotations[index]);
}

@Override
public int getParameterCount() {
    return _constructor.getParameterTypes().length;
}

@Override
public Class<?> getParameterClass(int index)
{
    Class<?>[] types = _constructor.getParameterTypes();
    return (index >= types.length) ? null : types[index];
}

@Override
public Type getParameterType(int index)
{
    Type[] types = _constructor.getGenericParameterTypes();
    return (index >= types.length) ? null : types[index];
}

/*
/**********************************************************
/* AnnotatedMember impl
/**********************************************************
 */

@Override
public Class<?> getDeclaringClass() { return _constructor.getDeclaringClass(); }

@Override
public Member getMember() { return _constructor; }
```

```
  /*
  /**********************************************************
  /* Extended API, specific annotations
  /**********************************************************
   */

  @Override
  public String toString() {
     return "[constructor for "+getName()+", annotations: "+_annotations+"]";
  }
}
package org.codehaus.jackson.map.introspect;

import java.lang.annotation.Annotation;
import java.lang.reflect.Field;
import java.lang.reflect.Member;
import java.lang.reflect.Type;

/**
* Object that represents non-static (and usually non-transient/volatile)
* fields of a class.
*
* @author tatu
*/
public final class AnnotatedField
  extends AnnotatedMember
{
  protected final Field _field;

  protected final AnnotationMap _annotations;

  /*
  /**********************************************************
  /* Life-cycle
  /**********************************************************
   */

  public AnnotatedField(Field field, AnnotationMap annMap)
  {
     _field = field;
     _annotations = annMap;
  }

  /**
   * Method called to override an annotation, usually due to a mix-in
   * annotation masking or overriding an annotation 'real' constructor
   * has.
   */
```

```
public void addOrOverride(Annotation a)
{
   _annotations.add(a);
}

/*
/**********************************************************
/* Annotated impl
/**********************************************************
 */

@Override
public Field getAnnotated() { return _field; }

@Override
public int getModifiers() { return _field.getModifiers(); }

@Override
public String getName() { return _field.getName(); }

@Override
public <A extends Annotation> A getAnnotation(Class<A> acls)
{
   return _annotations.get(acls);
}

@Override
public Type getGenericType() {
   return _field.getGenericType();
}

@Override
public Class<?> getRawType() {
   return _field.getType();
}

/*
/**********************************************************
/* AnnotatedMember impl
/**********************************************************
 */

@Override
public Class<?> getDeclaringClass() { return _field.getDeclaringClass(); }

@Override
public Member getMember() { return _field; }
```

```
  /*
  /**********************************************************
  /* Extended API, generic
  /**********************************************************
   */

  public String getFullName() {
     return getDeclaringClass().getName() + "#" + getName();
  }

  public int getAnnotationCount() { return _annotations.size(); }

  @Override
  public String toString()
  {
     return "[field "+getName()+", annotations: "+_annotations+"]";
  }
}
package org.codehaus.jackson.map.introspect;

import java.lang.reflect.Member;

import org.codehaus.jackson.map.util.ClassUtil;

/**
* Intermediate base class for annotated entities that are members of
* a class; fields, methods and constructors. This is a superset
* of things that can represent logical properties as it contains
* constructors in addition to fields and methods.
*
* @author tatu
* @since 1.5
*/
public abstract class AnnotatedMember extends Annotated
{
  protected AnnotatedMember() { super(); }

  public abstract Class<?> getDeclaringClass();

  public abstract Member getMember();

  /**
   * Method that can be called to modify access rights, by calling
   * {@link java.lang.reflect.AccessibleObject#setAccessible} on
   * the underlying annotated element.
   */
  public final void fixAccess() {
     ClassUtil.checkAndFixAccess(getMember());
```

```
  }
}
package org.codehaus.jackson.map.introspect;

import java.lang.annotation.Annotation;

import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.KeyDeserializer;
import org.codehaus.jackson.map.AnnotationIntrospector;
import org.codehaus.jackson.map.annotate.JsonSerialize.Inclusion;
import org.codehaus.jackson.map.annotate.JsonSerialize.Typing;

/**
* Dummy, "no-operation" implementation of {@link AnnotationIntrospector}.
* Can be used as is to suppress handling of annotations; or as a basis
* for simple complementary annotators
*/
public class NopAnnotationIntrospector
  extends AnnotationIntrospector
{
  /**
   * Static immutable and shareable instance that can be used as
   * "null" introspector: one that never finds any annotation
   * information.
   */
  public final static NopAnnotationIntrospector instance = new NopAnnotationIntrospector();

  /*
  /**********************************************************
  /* General annotation properties
  /**********************************************************
   */

  @Override
  public boolean isHandled(Annotation ann) {
     return false;
  }

  /*
  /**********************************************************
  /* General annotations
  /**********************************************************
   */

  @Override
  public String findEnumValue(Enum<?> value) {
```

```
    return null;
}

/*
/**********************************************************
/* General Class annotations
/**********************************************************
 */

@Override
public Boolean findCachability(AnnotatedClass ac) {
    return null;
}

@Override
public String findRootName(AnnotatedClass ac) {
    return null;
}

@Override
public String[] findPropertiesToIgnore(AnnotatedClass ac) {
    return null;
}

@Override
public Boolean findIgnoreUnknownProperties(AnnotatedClass ac) {
    return null;
}

/*
/**********************************************************
/* Property auto-detection
/**********************************************************
 */

@Override
public VisibilityChecker<?> findAutoDetectVisibility(AnnotatedClass ac, VisibilityChecker<?> checker) {
    return checker;
}

/*
/**********************************************************
/* General Method annotations
/**********************************************************
 */

@Override
public boolean isIgnorableConstructor(AnnotatedConstructor c) {
```

```
    return false;
}

@Override
public boolean isIgnorableMethod(AnnotatedMethod m) {
    return false;
}

/*
/**********************************************************
/* General field annotations
/**********************************************************
 */

@Override
public boolean isIgnorableField(AnnotatedField f) {
    return false;
}

/*
/**********************************************************
/* Serialization: general annotations
/**********************************************************
 */

@Override
public Object findSerializer(Annotated am, BeanProperty property) {
    return null;
}

@Override
public Inclusion findSerializationInclusion(Annotated a, Inclusion defValue) {
    return Inclusion.ALWAYS;
}

@Override
public Class<?> findSerializationType(Annotated a) {
    return null;
}

@Override
public Typing findSerializationTyping(Annotated a) {
    return null;
}

@Override
public Class<?>[] findSerializationViews(Annotated a) {
    return null;
```

```
}

/*
/**********************************************************
/* Serialization: class annotations
/**********************************************************
 */

@Override
public String[] findSerializationPropertyOrder(AnnotatedClass ac) {
   return null;
}

@Override
public Boolean findSerializationSortAlphabetically(AnnotatedClass ac) {
   return null;
}

/*
/**********************************************************
/* Serialization: method annotations
/**********************************************************
 */

@Override
public String findGettablePropertyName(AnnotatedMethod am) {
   return null;
}

@Override
public boolean hasAsValueAnnotation(AnnotatedMethod am) {
   return false;
}

@Override
public String findDeserializablePropertyName(AnnotatedField af) {
   return null;
}

@Override
public Class<?> findDeserializationContentType(Annotated am, JavaType t, String propName) {
   return null;
}

@Override
public Class<?> findDeserializationKeyType(Annotated am, JavaType t, String propName) {
   return null;
}
```

```
  @Override
  public Class<?> findDeserializationType(Annotated am, JavaType t, String propName) {
    return null;
  }

  @Override
  public Object findDeserializer(Annotated am, BeanProperty property) { return null; }

  @Override
  public Class<KeyDeserializer> findKeyDeserializer(Annotated am) { return null; }

  @Override
  public Class<JsonDeserializer<?>> findContentDeserializer(Annotated am) { return null; }


  @Override
  public String findPropertyNameForParam(AnnotatedParameter param) {
    return null;
  }

  @Override
  public String findSerializablePropertyName(AnnotatedField af) {
    return null;
  }

  @Override
  public String findSettablePropertyName(AnnotatedMethod am) {
    return null;
  }
}
package org.codehaus.jackson.map.introspect;

import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
import java.util.*;

import org.codehaus.jackson.map.AnnotationIntrospector;
import org.codehaus.jackson.map.ClassIntrospector;
import org.codehaus.jackson.map.DeserializationConfig;
import org.codehaus.jackson.map.MapperConfig;
import org.codehaus.jackson.map.SerializationConfig;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.type.JavaType;

public class BasicClassIntrospector
  extends ClassIntrospector<BasicBeanDescription>
```

```
{
  /**
   * Filter used to only include methods that have signature that is
   * compatible with "getters": take no arguments, are non-static,
   * and return something.
   */
  public static class GetterMethodFilter
     implements MethodFilter
  {
     public final static GetterMethodFilter instance = new GetterMethodFilter();

     private GetterMethodFilter() { }

     public boolean includeMethod(Method m)
     {
        return ClassUtil.hasGetterSignature(m);
     }
  }

  /**
   * Filter used to only include methods that have signature that is
   * compatible with "setters": take one and only argument and
   * are non-static.
   *<p>
   * Actually, also need to include 2-arg  methods to support
   * "any setters"; as well as 0-arg getters as long as they
   * return Collection or Map type.
   */
  public static class SetterMethodFilter
     implements MethodFilter
  {
     public final static SetterMethodFilter instance = new SetterMethodFilter();

     public boolean includeMethod(Method m)
     {
        // First: we can't use static methods
        if (Modifier.isStatic(m.getModifiers())) {
           return false;
        }
        int pcount = m.getParameterTypes().length;
        // Ok; multiple acceptable parameter counts:
        switch (pcount) {
        case 1:
           // Regular setters take just one param, so include:
           return true;
        case 2:
           /* 2-arg version only for "AnySetters"; they are not
            * auto-detected, and need to have an annotation.
```

```
             * However, due to annotation inheritance we do, we
             * don't yet know if sub-classes might have annotations...
             * so shouldn't leave out any methods quite yet.
             */
            //if (m.getAnnotation(JsonAnySetter.class) != null) { ... }

            return true;
        }
        return false;
    }
}

/**
 * Filter used if some getters (namely, once needed for "setterless
 * collection injection") are also needed, not just setters.
 */
public final static class SetterAndGetterMethodFilter
    extends SetterMethodFilter
{
    public final static SetterAndGetterMethodFilter instance = new SetterAndGetterMethodFilter();

    @Override
    public boolean includeMethod(Method m)
    {
        if (super.includeMethod(m)) {
            return true;
        }
        if (!ClassUtil.hasGetterSignature(m)) {
            return false;
        }
        // but furthermore, only accept Collections & Maps, for now
        Class<?> rt = m.getReturnType();
        if (Collection.class.isAssignableFrom(rt)
            || Map.class.isAssignableFrom(rt)) {
            return true;
        }
        return false;
    }
}

/*
/**********************************************************
/* Life cycle
/**********************************************************
 */

public final static BasicClassIntrospector instance = new BasicClassIntrospector();
```

```
public BasicClassIntrospector() { }

/*
/**********************************************************
/* Factory method impls
/**********************************************************
 */

@Override
public BasicBeanDescription forSerialization(SerializationConfig cfg,
       JavaType type, MixInResolver r)
{
   AnnotationIntrospector ai = cfg.getAnnotationIntrospector();
   AnnotatedClass ac = AnnotatedClass.construct(type.getRawClass(), ai, r);
   // False -> no need to collect ignorable member list
   ac.resolveMemberMethods(getSerializationMethodFilter(cfg), false);
   /* only the default constructor needed here (that's needed
    * in case we need to check default bean property values,
    * to omit them)
    */
   /* 31-Oct-2009, tatus: Actually, creator info will come in handy
    *   for resolving [JACKSON-170] as well
    */
   ac.resolveCreators(true);
   // False -> no need to collect ignorable field list
   ac.resolveFields(false);
   return new BasicBeanDescription(type, ac, ai);
}

@Override
public BasicBeanDescription forDeserialization(DeserializationConfig cfg,
                                   JavaType type,
                                   MixInResolver r)
{
   AnnotationIntrospector ai = cfg.getAnnotationIntrospector();
   AnnotatedClass ac = AnnotatedClass.construct(type.getRawClass(), ai, r);
   // everything needed for deserialization, including ignored methods
   ac.resolveMemberMethods(getDeserializationMethodFilter(cfg), true);
   // include all kinds of creator methods:
   ac.resolveCreators(true);
   // yes, we need info on ignored fields as well
   ac.resolveFields(true);
   return new BasicBeanDescription(type, ac, ai);
}

@Override
public BasicBeanDescription forCreation(DeserializationConfig cfg,
       JavaType type, MixInResolver r)
```

```
{
   AnnotationIntrospector ai = cfg.getAnnotationIntrospector();
   AnnotatedClass ac = AnnotatedClass.construct(type.getRawClass(), ai, r);
   ac.resolveCreators(true);
   return new BasicBeanDescription(type, ac, ai);
}

@Override
public BasicBeanDescription forClassAnnotations(MapperConfig<?> cfg,
      Class<?> c, MixInResolver r)
{
   AnnotationIntrospector ai = cfg.getAnnotationIntrospector();
   AnnotatedClass ac = AnnotatedClass.construct(c, ai, r);
   return new BasicBeanDescription(TypeFactory.type(c), ac, ai);
}

@Override
public BasicBeanDescription forDirectClassAnnotations(MapperConfig<?> cfg,
      Class<?> c, MixInResolver r)
{
   AnnotationIntrospector ai = cfg.getAnnotationIntrospector();
   AnnotatedClass ac = AnnotatedClass.constructWithoutSuperTypes(c, ai, r);
   return new BasicBeanDescription(TypeFactory.type(c), ac, ai);
}

/*
/**********************************************************
/* Overridable helper methods
/**********************************************************
 */

/**
 * Helper method for getting access to filter that only guarantees
 * that methods used for serialization are to be included.
 */
protected MethodFilter getSerializationMethodFilter(SerializationConfig cfg)
{
 return GetterMethodFilter.instance;
}

/**
 * Helper method for getting access to filter that only guarantees
 * that methods used for deserialization are to be included.
 */
protected MethodFilter getDeserializationMethodFilter(DeserializationConfig cfg)
{
   /* [JACKSON-88]: may also need to include getters (at least for
    * Collection and Map types)
```

```
         */
        if (cfg.isEnabled(DeserializationConfig.Feature.USE_GETTERS_AS_SETTERS)) {
            return SetterAndGetterMethodFilter.instance;

        }
     return SetterMethodFilter.instance;
    }
}
package org.codehaus.jackson.map.introspect;

import java.lang.annotation.Annotation;
import java.lang.reflect.AnnotatedElement;
import java.lang.reflect.Member;
import java.lang.reflect.Type;

import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.type.JavaType;

/**
* Object that represents method parameters, mostly so that associated
* annotations can be processed conveniently. Note that many of accessors
* can not return meaningful values since parameters do not have stand-alone
* JDK objects associated; so access should mostly be limited to checking
* annotation values which are properly aggregated and included.
*<p>
* Note: as of version 1.7, this type extends {@link AnnotatedMember}, since
* it behaves like a member for the most part, but earlier it just extended
* {@link Annotated}
*/
public final class AnnotatedParameter
    extends AnnotatedMember
{
    /**
     * Member (method, constructor) that this parameter belongs to
     *
     * @since 1.7
     */
    protected final AnnotatedMember _owner;

    /**
     * JDK type of the parameter, possibly contains generic type information
     */
    protected final Type _type;

    protected final AnnotationMap _annotations;

    /*
    /**********************************************************
```

```
/* Life-cycle
/**********************************************************
 */

public AnnotatedParameter(AnnotatedMember owner, Type type,  AnnotationMap ann)
{
   _owner = owner;
   _type = type;
   _annotations = ann;
}

public void addOrOverride(Annotation a)
{
   _annotations.add(a);
}

/*
/**********************************************************
/* Annotated impl
/**********************************************************
 */

/**
 * Since there is no matching JDK element, this method will
 * always return null
 */
@Override
public AnnotatedElement getAnnotated() { return null; }

/**
 * Returns modifiers of the constructor, as parameters do not
 * have independent modifiers.
 */
@Override
public int getModifiers() { return _owner.getModifiers(); }

/**
 * Parameters have no names in bytecode (unlike in source code),
 * will always return empty String ("").
 */
@Override
public String getName() { return ""; }

/**
 * Accessor for annotations; all annotations associated with parameters
 * are properly passed and accessible.
 */
@Override
```

```
  public <A extends Annotation> A getAnnotation(Class<A> acls)
  {
     return _annotations.get(acls);
  }

  @Override
  public Type getGenericType() {
     return _type;
  }

  @Override
  public Class<?> getRawType() {
     JavaType t = TypeFactory.type(_type);
     return t.getRawClass();
  }

  /*
  /**********************************************************
  /* AnnotatedMember extras
  /**********************************************************
   */

  @Override
  public Class<?> getDeclaringClass() {
     return _owner.getDeclaringClass();
  }

  @Override
  public Member getMember() {
     /* This is bit tricky: since there is no JDK equivalent; can either
      * return null or owner... let's do latter, for now.
      */
     return _owner.getMember();
  }

  /*
  /**********************************************************
  /* Extended API
  /**********************************************************
   */

  public Type getParameterType() { return _type; }
}
package org.codehaus.jackson.map;

import java.io.IOException;

import org.codehaus.jackson.*;
```

```
/**
* Abstract class that defines API used by {@link ObjectMapper} (and
* other chained {@link JsonDeserializer}s too) to deserialize Objects of
* arbitrary types from JSON, using provided {@link JsonParser}.
*/
public abstract class JsonDeserializer<T>
{
  /**
   * Method that can be called to ask implementation to deserialize
   * json content into the value type this serializer handles.
   * Returned instance is to be constructed by method itself.
   *<p>
   * Pre-condition for this method is that the parser points to the
   * first event that is part of value to deserializer (and which
   * is never Json 'null' literal, more on this below): for simple
   * types it may be the only value; and for structured types the
   * Object start marker.
   * Post-condition is that the parser will point to the last
   * event that is part of deserialized value (or in case deserialization
   * fails, event that was not recognized or usable, which may be
   * the same event as the one it pointed to upon call).
   *<p>
   * Note that this method is never called for JSON null literal,
   * and thus deserializers need (and should) not check for it.
   *
   * @param jp Parsed used for reading Json content
   * @param ctxt Context that can be used to access information about
   *   this deserialization activity.
   *
   * @return Deserializer value
   */
  public abstract T deserialize(JsonParser jp, DeserializationContext ctxt)
     throws IOException, JsonProcessingException;

  /**
   * Alternate deserialization method (compared to the most commonly
   * used, {@link #deserialize(JsonParser, DeserializationContext)}),
   * which takes in initialized value instance, to be
   * configured and/or populated by deserializer.
   * Method is not necessarily used for all supported types; most commonly
   * it is used
   * for Collections and Maps.
   *<p>
   * Default implementation just throws
   * {@link UnsupportedOperationException}, to indicate that types
   * that do not explicitly add support do not expect to get the call.
   */
```

```
public T deserialize(JsonParser jp, DeserializationContext ctxt,
                T intoValue)
   throws IOException, JsonProcessingException
{
   throw new UnsupportedOperationException();
}

/**
 * Deserialization called when type being deserialized is defined to
 * contain additional type identifier, to allow for correctly
 * instantiating correct subtype. This can be due to annotation on
 * type (or its supertype), or due to global settings without
 * annotations.
 *<p>
 * Default implementation may work for some types, but ideally subclasses
 * should not rely on current default implementation.
 * Implementation is mostly provided to avoid compilation errors with older
 * code.
 *
 * @param typeDeserializer Deserializer to use for handling type information
 *
 * @since 1.5
 */
@SuppressWarnings("unchecked")
public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
       TypeDeserializer typeDeserializer)
   throws IOException, JsonProcessingException
{
   // We could try calling
   return (T) typeDeserializer.deserializeTypedFromAny(jp, ctxt);
}

/**
 * Method that can be called to determine value to be used for
 * representing null values (values deserialized when Json token
 * is {@link JsonToken#VALUE_NULL}). Usually this is simply
 * Java null, but for some types (primitives) it may be
 * necessary to use actual values.
 *<p>
 * Note that deserializers are allowed to call this just once and
 * then reuse returned value; that is, method is not guaranteed to
 * be called once for each conversion.
 *<p>
 * Default implementation simply returns null.
 */
public T getNullValue() { return null; }

/*
```

```
  /**********************************************************
  /* Helper classes
  /**********************************************************
   */

  /**
   * This marker class is only to be used with annotations, to
   * indicate that <b>no deserializer is configured</b>.
   *<p>
   * Specifically, this class is to be used as the marker for
   * annotation {@link org.codehaus.jackson.map.annotate.JsonDeserialize}
   */
  public abstract static class None
      extends JsonDeserializer<Object> { }
}
package org.codehaus.jackson.map;

import java.io.IOException;
import java.util.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.util.ArrayBuilders;
import org.codehaus.jackson.map.util.ObjectBuffer;
import org.codehaus.jackson.node.JsonNodeFactory;
import org.codehaus.jackson.type.JavaType;

/**
* Context for deserialization process. Used to allow passing in configuration
* settings and reusable temporary objects (scrap arrays, containers).
*/
public abstract class DeserializationContext
{
  protected final DeserializationConfig _config;

  /**
   * @since 1.7
   */
  protected final int _featureFlags;

  /*
  /**********************************************************
  /* Life-cycle
  /**********************************************************
   */

  protected DeserializationContext(DeserializationConfig config)
  {
      _config = config;
```

```
    _featureFlags = config._featureFlags;
}

/*
/**********************************************************
/* Configuration methods
/**********************************************************
 */

/**
 * Method for accessing configuration setting object for
 * currently active deserialization.
 */
public DeserializationConfig getConfig() { return _config; }

/**
 * Returns provider that can be used for dynamically locating
 * other deserializers during runtime.
 *
 * @since 1.5
 */
public DeserializerProvider getDeserializerProvider() {
    // will be overridden by impl class
    return null;
}

/**
 * Convenience method for checking whether specified on/off
 * feature is enabled
 */
public boolean isEnabled(DeserializationConfig.Feature feat) {
    /* 03-Dec-2010, tatu: minor shortcut; since this is called quite often,
     *   let's use a local copy of feature settings:
     */
    return (_featureFlags & feat.getMask()) != 0;
}

/**
 * Convenience method for accessing the default Base64 encoding
 * used for decoding base64 encoded binary content.
 * Same as calling:
 *<pre>
 *  getConfig().getBase64Variant();
 *</pre>
 */
public Base64Variant getBase64Variant() {
    return _config.getBase64Variant();
}
```

```
/**
 * Accessor for getting access to the underlying JSON parser used
 * for deserialization.
 */
public abstract JsonParser getParser();

public final JsonNodeFactory getNodeFactory() {
   return _config.getNodeFactory();
}

/*
/**********************************************************
/* Methods for accessing reusable/recyclable helper objects
/**********************************************************
 */

/**
 * Method that can be used to get access to a reusable ObjectBuffer,
 * useful for efficiently constructing Object arrays and Lists.
 * Note that leased buffers should be returned once deserializer
 * is done, to allow for reuse during same round of deserialization.
 */
public abstract ObjectBuffer leaseObjectBuffer();

/**
 * Method to call to return object buffer previously leased with
 * {@link #leaseObjectBuffer}.
 *
 * @param buf Returned object buffer
 */
public abstract void returnObjectBuffer(ObjectBuffer buf);

/**
 * Method for accessing object useful for building arrays of
 * primitive types (such as int[]).
 */
public abstract ArrayBuilders getArrayBuilders();

/*
/**********************************************************
/* Parsing methods that may use reusable/-cyclable objects
/**********************************************************
 */

/**
 * Convenience method for parsing a Date from given String, using
 * currently configured date format (accessed using
```

```
     * {@link DeserializationConfig#getDateFormat()}).
     *<p>
     * Implementation will handle thread-safety issues related to
     * date formats such that first time this method is called,
     * date format is cloned, and cloned instance will be retained
     * for use during this deserialization round.
     */
    public abstract java.util.Date parseDate(String dateStr)
        throws IllegalArgumentException;

    /**
     * Convenience method for constructing Calendar instance set
     * to specified time, to be modified and used by caller.
     */
    public abstract Calendar constructCalendar(Date d);

    /*
    /**********************************************************
    /* Methods for problem handling, reporting
    /**********************************************************
     */

    /**
     * Method deserializers can call to inform configured {@link DeserializationProblemHandler}s
     * of an unrecognized property.
     *
     * @return True if there was a configured problem handler that was able to handle the
     *   proble
     *
     * @since 1.5
     */
    public abstract boolean handleUnknownProperty(JsonParser jp, JsonDeserializer<?> deser, Object
instanceOrClass, String propName)
        throws IOException, JsonProcessingException;

    /**
     * Helper method for constructing generic mapping exception for specified type
     */
    public abstract JsonMappingException mappingException(Class<?> targetClass);

    /**
     * Helper method for constructing generic mapping exception with specified
     * message and current location information
     *
     * @since 1.7
     */
    public JsonMappingException mappingException(String message)
    {
```

```
    return JsonMappingException.from(getParser(), message);
}

/**
 * Helper method for constructing instantiation exception for specified type,
 * to indicate problem with physically constructing instance of
 * specified class (missing constructor, exception from constructor)
 */
public abstract JsonMappingException instantiationException(Class<?> instClass, Exception e);

public abstract JsonMappingException instantiationException(Class<?> instClass, String msg);

/**
 * Helper method for constructing exception to indicate that input JSON
 * String was not in recognized format for deserializing into given type.
 */
public abstract JsonMappingException weirdStringException(Class<?> instClass, String msg);

/**
 * Helper method for constructing exception to indicate that input JSON
 * Number was not suitable for deserializing into given type.
 */
public abstract JsonMappingException weirdNumberException(Class<?> instClass, String msg);

/**
 * Helper method for constructing exception to indicate that given JSON
 * Object field name was not in format to be able to deserialize specified
 * key type.
 */
public abstract JsonMappingException weirdKeyException(Class<?> keyClass, String keyValue, String msg);

/**
 * Helper method for indicating that the current token was expected to be another
 * token.
 */
public abstract JsonMappingException wrongTokenException(JsonParser jp, JsonToken expToken, String msg);

/**
 * Helper method for constructing exception to indicate that JSON Object
 * field name did not map to a known property of type being
 * deserialized.
 *
 * @param instanceOrClass Either value being populated (if one has been
 *   instantiated), or Class that indicates type that would be (or
 *   have been) instantiated
 */
public abstract JsonMappingException unknownFieldException(Object instanceOrClass, String fieldName);
```

```
  /**
   * Helper method for constructing exception to indicate that given
   * type id (parsed from JSON) could not be converted to a Java type.
   *
   * @since 1.5
   */
  public abstract JsonMappingException unknownTypeException(JavaType baseType, String id);
}
/**
* Contains implementation classes of serialization part of
* data binding.
*/
package org.codehaus.jackson.map.ser;
package org.codehaus.jackson.map.ser;

import org.codehaus.jackson.map.TypeSerializer;

/**
* Intermediate base class for types that contain element(s) of
* other types. Used for example for List, Map, Object array and
* Iterator serializers.
*
* @since 1.5
*/
public abstract class ContainerSerializerBase<T>
  extends SerializerBase<T>
{
  /*
  /**********************************************************
  /* Construction, initialization
  /**********************************************************
   */

  protected ContainerSerializerBase(Class<T> t) {
     super(t);
  }

  /**
   * Alternate constructor that is (alas!) needed to work
   * around kinks of generic type handling
   *
   * @param t
   */
  protected ContainerSerializerBase(Class<?> t, boolean dummy) {
     super(t, dummy);
  }

  /**
```

```
     * Factory(-like) method that can be used to construct a new container
     * serializer that uses specified {@link TypeSerializer} for decorating
     * contained values with additional type information.
     *
     * @param vts Type serializer to use for contained values; can be null,
     *    in which case 'this' serializer is returned as is
     * @return Serializer instance that uses given type serializer for values if
     *    that is possible (or if not, just 'this' serializer)
     */
    public ContainerSerializerBase<?> withValueTypeSerializer(TypeSerializer vts) {
        if (vts == null) return this;
        return _withValueTypeSerializer(vts);
    }

    public abstract ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts);
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.Type;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;

/**
* Simple general purpose serializer, useful for any
* type for which {@link Object#toString} returns the desired Json
* value.
*/
@JacksonStdImpl
public final class ToStringSerializer
    extends SerializerBase<Object>
{
    /**
     * Singleton instance to use.
     */
    public final static ToStringSerializer instance = new ToStringSerializer();

    /**
     *<p>
     * Note: usually you should NOT create new instances, but instead use
     * {@link #instance} which is stateless and fully thread-safe. However,
     * there are cases where constructor is needed; for example,
     * when using explicit serializer annotations like
```

```
     * {@link org.codehaus.jackson.map.annotate.JsonSerialize#using}.
     */
    public ToStringSerializer() { super(Object.class); }

    @Override
    public void serialize(Object value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeString(value.toString());
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
        throws JsonMappingException
    {
        return createSchemaNode("string", true);
    }

}
package org.codehaus.jackson.map.ser;

import java.lang.reflect.Type;
import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;

/**
* This is a simple dummy serializer that will just output literal
* JSON null value whenever serialization is requested.
* Used as the default "null serializer" (which is used for serializing
* null object references unless overridden), as well as for some
* more exotic types (java.lang.Void).
*/
@JacksonStdImpl
public class NullSerializer
    extends SerializerBase<Object>
{
    public final static NullSerializer instance = new NullSerializer();

    private NullSerializer() { super(Object.class); }

    @Override
    public void serialize(Object value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeNull();
```

```
  }

  @Override
  public JsonNode getSchema(SerializerProvider provider, Type typeHint)
     throws JsonMappingException
  {
     return createSchemaNode("null");
  }
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.Type;
import java.util.*;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.type.JavaType;

/**
* Standard serializer implementation for serializing {link java.util.Map} types.
*<p>
* Note: about the only configurable setting currently is ability to filter out
* entries with specified names.
*/
@JacksonStdImpl
public class MapSerializer
  extends ContainerSerializerBase<Map<?,?>>
  implements ResolvableSerializer
{
  protected final static JavaType UNSPECIFIED_TYPE = TypeFactory.fastSimpleType(Object.class);

  /**
   * Map-valued property being serialized with this instance
   *
   * @since 1.7
   */
  protected final BeanProperty _property;

  /**
   * Set of entries to omit during serialization, if any
   */
  protected final HashSet<String> _ignoredEntries;
```

```
/**
 * Whether static types should be used for serialization of values
 * or not (if not, dynamic runtime type is used)
 */
protected final boolean _valueTypeIsStatic;

/**
 * Declared type of keys
 *
 * @since 1.7
 */
protected final JavaType _keyType;

/**
 * Declared type of contained values
 */
protected final JavaType _valueType;

/**
 * Key serializer to use, if it can be statically determined
 *
 * @since 1.7
 */
protected JsonSerializer<Object> _keySerializer;

/**
 * Value serializer to use, if it can be statically determined
 *
 * @since 1.5
 */
protected JsonSerializer<Object> _valueSerializer;

/**
 * Type identifier serializer used for values, if any.
 */
protected final TypeSerializer _valueTypeSerializer;

protected MapSerializer() {
    this((HashSet<String>)null, null, null, false, null, null, null);
}

/**
 * Legacy constructor (as of 1.7)
 *
 * @deprecated Use variant that takes Key type and property information
 */
@Deprecated
```

```
protected MapSerializer(HashSet<String> ignoredEntries,
        JavaType valueType, boolean valueTypeIsStatic,
        TypeSerializer vts)
{
    this(ignoredEntries, UNSPECIFIED_TYPE, valueType, valueTypeIsStatic, vts, null, null);
}

protected MapSerializer(HashSet<String> ignoredEntries,
        JavaType keyType, JavaType valueType, boolean valueTypeIsStatic,
        TypeSerializer vts, JsonSerializer<Object> keySerializer,
        BeanProperty property)
{
    super(Map.class, false);
    _property = property;
    _ignoredEntries = ignoredEntries;
    _keyType = keyType;
    _valueType = valueType;
    _valueTypeIsStatic = valueTypeIsStatic;
    _valueTypeSerializer = vts;
    _keySerializer = keySerializer;
}

@Override
public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts)
{
    MapSerializer ms = new MapSerializer(_ignoredEntries, _keyType, _valueType, _valueTypeIsStatic, vts,
            _keySerializer, _property);
    if (_valueSerializer != null) {
        ms._valueSerializer = _valueSerializer;
    }
    return ms;
}

/**
 * Factory method used to construct Map serializers.
 *
 * @param ignoredList Array of entry names that are to be filtered on
 *    serialization; null if none
 * @param mapType Declared type information (needed for static typing)
 * @param staticValueType Whether static typing should be used for the
 *    Map (which includes its contents)
 * @param vts Type serializer to use for map entry values, if any
 */
public static MapSerializer construct(String[] ignoredList, JavaType mapType,
        boolean staticValueType, TypeSerializer vts, BeanProperty property)
{
    HashSet<String> ignoredEntries = toSet(ignoredList);
    JavaType keyType, valueType;
```

```
    if (mapType == null) {
        keyType = valueType = UNSPECIFIED_TYPE;
    } else {
        keyType = mapType.getKeyType();
        valueType = mapType.getContentType();
    }
    // If value type is final, it's same as forcing static value typing:
    if (!staticValueType) {
        staticValueType = (valueType != null && valueType.isFinal());
    }
    return new MapSerializer(ignoredEntries, keyType, valueType, staticValueType, vts,
            null, property);
}

private static HashSet<String> toSet(String[] ignoredEntries) {
    if (ignoredEntries == null || ignoredEntries.length == 0) {
        return null;
    }
    HashSet<String> result = new HashSet<String>(ignoredEntries.length);
    for (String prop : ignoredEntries) {
        result.add(prop);
    }
    return result;
}

/*
/**********************************************************
/* JsonSerializer implementation
/**********************************************************
 */

@Override
public void serialize(Map<?,?> value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
{
    jgen.writeStartObject();
    if (!value.isEmpty()) {
        if (_valueSerializer != null) {
            serializeFieldsUsing(value, jgen, provider, _valueSerializer);
        } else {
            serializeFields(value, jgen, provider);
        }
    }
    jgen.writeEndObject();
}

@Override
```

```
public void serializeWithType(Map<?,?> value, JsonGenerator jgen, SerializerProvider provider,
        TypeSerializer typeSer)
    throws IOException, JsonGenerationException
{
    typeSer.writeTypePrefixForObject(value, jgen);
    if (!value.isEmpty()) {
        if (_valueSerializer != null) {
            serializeFieldsUsing(value, jgen, provider, _valueSerializer);
        } else {
            serializeFields(value, jgen, provider);
        }
    }
    typeSer.writeTypeSuffixForObject(value, jgen);
}

/*
/**********************************************************
/* JsonSerializer implementation
/**********************************************************
 */

/**
 * Method called to serialize fields, when the value type is not statically known.
 */
protected void serializeFields(Map<?,?> value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
{
    // If value type needs polymorphic type handling, some more work needed:
    if (_valueTypeSerializer != null) {
        serializeTypedFields(value, jgen, provider);
        return;
    }
    final JsonSerializer<Object> keySerializer = _keySerializer;

    JsonSerializer<Object> prevValueSerializer = null;
    Class<?> prevValueClass = null;
    final HashSet<String> ignored = _ignoredEntries;
    final boolean skipNulls = !provider.isEnabled(SerializationConfig.Feature.WRITE_NULL_MAP_VALUES);

    for (Map.Entry<?,?> entry : value.entrySet()) {
        Object valueElem = entry.getValue();
        // First, serialize key
        Object keyElem = entry.getKey();
        if (keyElem == null) {
            provider.getNullKeySerializer().serialize(null, jgen, provider);
        } else {
            // [JACKSON-314] skip entries with null values?
            if (skipNulls && valueElem == null) continue;
```

```
            // One twist: is entry ignorable? If so, skip
            if (ignored != null && ignored.contains(keyElem)) continue;
            keySerializer.serialize(keyElem, jgen, provider);
        }

        // And then value
        if (valueElem == null) {
            provider.defaultSerializeNull(jgen);
        } else {
            Class<?> cc = valueElem.getClass();
            JsonSerializer<Object> currSerializer;
            if (cc == prevValueClass) {
                currSerializer = prevValueSerializer;
            } else {
                currSerializer = provider.findValueSerializer(cc, _property);
                prevValueSerializer = currSerializer;
                prevValueClass = cc;
            }
            try {
                currSerializer.serialize(valueElem, jgen, provider);
            } catch (Exception e) {
                // [JACKSON-55] Need to add reference information
                String keyDesc = ""+keyElem;
                wrapAndThrow(provider, e, value, keyDesc);
            }
        }
    }
}

/**
 * Method called to serialize fields, when the value type is statically known,
 * so that value serializer is passed and does not need to be fetched from
 * provider.
 */
protected void serializeFieldsUsing(Map<?,?> value, JsonGenerator jgen, SerializerProvider provider,
        JsonSerializer<Object> ser)
        throws IOException, JsonGenerationException
{
    final JsonSerializer<Object> keySerializer = _keySerializer;
    final HashSet<String> ignored = _ignoredEntries;
    final TypeSerializer typeSer = _valueTypeSerializer;
    final boolean skipNulls = !provider.isEnabled(SerializationConfig.Feature.WRITE_NULL_MAP_VALUES);

    for (Map.Entry<?,?> entry : value.entrySet()) {
        Object valueElem = entry.getValue();
        Object keyElem = entry.getKey();
        if (keyElem == null) {
            provider.getNullKeySerializer().serialize(null, jgen, provider);
```

```
        } else {
            // [JACKSON-314] also may need to skip entries with null values
            if (skipNulls && valueElem == null) continue;
            if (ignored != null && ignored.contains(keyElem)) continue;
            keySerializer.serialize(keyElem, jgen, provider);
        }
        if (valueElem == null) {
            provider.defaultSerializeNull(jgen);
        } else {
            try {
                if (typeSer == null) {
                    ser.serialize(valueElem, jgen, provider);
                } else {
                    ser.serializeWithType(valueElem, jgen, provider, typeSer);
                }
            } catch (Exception e) {
                // [JACKSON-55] Need to add reference information
                String keyDesc = ""+keyElem;
                wrapAndThrow(provider, e, value, keyDesc);
            }
        }
    }
}

protected void serializeTypedFields(Map<?,?> value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
{
    final JsonSerializer<Object> keySerializer = _keySerializer;
    JsonSerializer<Object> prevValueSerializer = null;
    Class<?> prevValueClass = null;
    final HashSet<String> ignored = _ignoredEntries;
    final boolean skipNulls = !provider.isEnabled(SerializationConfig.Feature.WRITE_NULL_MAP_VALUES);

    for (Map.Entry<?,?> entry : value.entrySet()) {
        Object valueElem = entry.getValue();
        // First, serialize key
        Object keyElem = entry.getKey();
        if (keyElem == null) {
            provider.getNullKeySerializer().serialize(null, jgen, provider);
        } else {
            // [JACKSON-314] also may need to skip entries with null values
            if (skipNulls && valueElem == null) continue;
            // One twist: is entry ignorable? If so, skip
            if (ignored != null && ignored.contains(keyElem)) continue;
            keySerializer.serialize(keyElem, jgen, provider);
        }

        // And then value
```

```
            if (valueElem == null) {
                provider.defaultSerializeNull(jgen);
            } else {
                Class<?> cc = valueElem.getClass();
                JsonSerializer<Object> currSerializer;
                if (cc == prevValueClass) {
                    currSerializer = prevValueSerializer;
                } else {
                    currSerializer = provider.findValueSerializer(cc, _property);
                    prevValueSerializer = currSerializer;
                    prevValueClass = cc;
                }
                try {
                    currSerializer.serializeWithType(valueElem, jgen, provider, _valueTypeSerializer);
                } catch (Exception e) {
                    // [JACKSON-55] Need to add reference information
                    String keyDesc = ""+keyElem;
                    wrapAndThrow(provider, e, value, keyDesc);
                }
            }
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        ObjectNode o = createSchemaNode("object", true);
        //(ryan) even though it's possible to statically determine the "value" type of the map,
        // there's no way to statically determine the keys, so the "Entries" can't be determined.
        return o;
    }

    /**
     * Need to get callback to resolve value serializer, if static typing
     * is used (either being forced, or because value type is final)
     */
    public void resolve(SerializerProvider provider)
        throws JsonMappingException
    {
        if (_valueTypeIsStatic) {
            _valueSerializer = provider.findValueSerializer(_valueType, _property);
        }
        /* 10-Dec-2010, tatu: Let's also fetch key serializer; and always assume we'll
         *   do that just using static type information
         */
        _keySerializer = provider.getKeySerializer(_keyType, _property);
    }
}
```

```
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.Type;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Special bogus "serializer" that will throw
* {@link JsonGenerationException} if its {@link #serialize}
* gets invoeked. Most commonly registered as handler for unknown types,
* as well as for catching unintended usage (like trying to use null
* as Map/Object key).
*/
public final class FailingSerializer
  extends SerializerBase<Object>
{
  final String _msg;

  public FailingSerializer(String msg) {
    super(Object.class);
    _msg = msg;
  }

  @Override
  public void serialize(Object value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
  {
    throw new JsonGenerationException(_msg);
  }

  @Override
  public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    throws JsonMappingException
  {
    return null;
  }
}
/**
* Contains implementation classes of serialization part of
* data binding.
*/
package org.codehaus.jackson.map.ser.impl;
```

```
package org.codehaus.jackson.map.ser.impl;

import java.io.IOException;
import java.lang.reflect.Type;
import java.net.InetAddress;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.ser.ScalarSerializerBase;

/**
* Simple serializer for {@link java.net.InetAddress}. Main complexity is
* with registration, since same serializer is to be used for sub-classes.
*
* @since 1.8
*/
public class InetAddressSerializer
  extends ScalarSerializerBase<InetAddress>
{
  public final static InetAddressSerializer instance = new InetAddressSerializer();

  public InetAddressSerializer() { super(InetAddress.class); }

  @Override
  public void serialize(InetAddress value, JsonGenerator jgen, SerializerProvider provider)
     throws IOException, JsonGenerationException
  {
     // Ok: get textual description; choose "more specific" part
     String str = value.toString().trim();
     int ix = str.indexOf('/');
     if (ix >= 0) {
        if (ix == 0) { // missing host name; use address
           str = str.substring(1);
        } else { // otherwise use name
           str = str.substring(0, ix);
        }
     }
     jgen.writeString(str);
  }

  @Override
  public JsonNode getSchema(SerializerProvider provider, Type typeHint) {
     return createSchemaNode("string", true);
  }
}
```

```
package org.codehaus.jackson.map.ser.impl;

import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Type;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.ResolvableSerializer;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.TypeSerializer;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.ser.ArraySerializers;
import org.codehaus.jackson.map.ser.ContainerSerializerBase;
import org.codehaus.jackson.map.type.ArrayType;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.schema.JsonSchema;
import org.codehaus.jackson.schema.SchemaAware;
import org.codehaus.jackson.type.JavaType;

/**
* Generic serializer for Object arrays (<code>Object[]</code>).
*/
@JacksonStdImpl
public class ObjectArraySerializer
  extends ArraySerializers.AsArraySerializer<Object[]>
  implements ResolvableSerializer
{
  /**
   * Whether we are using static typing (using declared types, ignoring
   * runtime type) or not for elements.
   */
  protected final boolean _staticTyping;

  /**
   * Declared type of element entries
   */
  protected final JavaType _elementType;

  /**
   * Value serializer to use, if it can be statically determined.
   *
   * @since 1.5
```

```
 */
protected JsonSerializer<Object> _elementSerializer;

/**
 * If element type can not be statically determined, mapping from
 * runtime type to serializer is handled using this object
 *
 * @since 1.7
 */
protected PropertySerializerMap _dynamicSerializers;

public ObjectArraySerializer(JavaType elemType, boolean staticTyping,
        TypeSerializer vts, BeanProperty property)
{
    super(Object[].class, vts, property);
    _elementType = elemType;
    _staticTyping = staticTyping;
    _dynamicSerializers = PropertySerializerMap.emptyMap();
}

@Override
public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts)
{
    return new ObjectArraySerializer(_elementType, _staticTyping, vts, _property);
}

@Override
public void serializeContents(Object[] value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
{
    final int len = value.length;
    if (len == 0) {
        return;
    }
    if (_elementSerializer != null) {
        serializeContentsUsing(value, jgen, provider, _elementSerializer);
        return;
    }
    if (_valueTypeSerializer != null) {
        serializeTypedContents(value, jgen, provider);
        return;
    }
    int i = 0;
    Object elem = null;
    try {
        PropertySerializerMap serializers = _dynamicSerializers;
        for (; i < len; ++i) {
            elem = value[i];
```

```
            if (elem == null) {
                provider.defaultSerializeNull(jgen);
                continue;
            }
            Class<?> cc = elem.getClass();
            JsonSerializer<Object> serializer = serializers.serializerFor(cc);
            if (serializer == null) {
                // To fix [JACKSON-508]
                if (_elementType.hasGenericTypes()) {
                    serializer = _findAndAddDynamic(serializers, _elementType.forcedNarrowBy(cc), provider);
                } else {
                    serializer = _findAndAddDynamic(serializers, cc, provider);
                }
            }
            serializer.serialize(elem, jgen, provider);
        }
    } catch (IOException ioe) {
        throw ioe;
    } catch (Exception e) {
        // [JACKSON-55] Need to add reference information
        /* 05-Mar-2009, tatu: But one nasty edge is when we get
         *   StackOverflow: usually due to infinite loop. But that gets
         *   hidden within an InvocationTargetException...
         */
        Throwable t = e;
        while (t instanceof InvocationTargetException && t.getCause() != null) {
            t = t.getCause();
        }
        if (t instanceof Error) {
            throw (Error) t;
        }
        throw JsonMappingException.wrapWithPath(t, elem, i);
    }
}

public void serializeContentsUsing(Object[] value, JsonGenerator jgen, SerializerProvider provider,
        JsonSerializer<Object> ser)
    throws IOException, JsonGenerationException
{
    final int len = value.length;
    final TypeSerializer typeSer = _valueTypeSerializer;

    int i = 0;
    Object elem = null;
    try {
        for (; i < len; ++i) {
            elem = value[i];
            if (elem == null) {
```

```
                provider.defaultSerializeNull(jgen);
                continue;
            }
            if (typeSer == null) {
                ser.serialize(elem, jgen, provider);
            } else {
                ser.serializeWithType(elem, jgen, provider, typeSer);
            }
        }
    } catch (IOException ioe) {
        throw ioe;
    } catch (Exception e) {
        Throwable t = e;
        while (t instanceof InvocationTargetException && t.getCause() != null) {
            t = t.getCause();
        }
        if (t instanceof Error) {
            throw (Error) t;
        }
        throw JsonMappingException.wrapWithPath(t, elem, i);
    }
}

public void serializeTypedContents(Object[] value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
{
    final int len = value.length;
    final TypeSerializer typeSer = _valueTypeSerializer;
    int i = 0;
    Object elem = null;
    try {
        PropertySerializerMap serializers = _dynamicSerializers;
        for (; i < len; ++i) {
            elem = value[i];
            if (elem == null) {
                provider.defaultSerializeNull(jgen);
                continue;
            }
            Class<?> cc = elem.getClass();
            JsonSerializer<Object> serializer = serializers.serializerFor(cc);
            if (serializer == null) {
                serializer = _findAndAddDynamic(serializers, cc, provider);
            }
            serializer.serializeWithType(elem, jgen, provider, typeSer);
        }
    } catch (IOException ioe) {
        throw ioe;
    } catch (Exception e) {
```

```
            Throwable t = e;
            while (t instanceof InvocationTargetException && t.getCause() != null) {
                t = t.getCause();
            }
            if (t instanceof Error) {
                throw (Error) t;
            }
            throw JsonMappingException.wrapWithPath(t, elem, i);
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
        throws JsonMappingException
    {
        ObjectNode o = createSchemaNode("array", true);
        if (typeHint != null) {
            JavaType javaType = TypeFactory.type(typeHint);
            if (javaType.isArrayType()) {
                Class<?> componentType = ((ArrayType) javaType).getContentType().getRawClass();
                // 15-Oct-2010, tatu: We can't serialize plain Object.class; but what should it produce here? Untyped?
                if (componentType == Object.class) {
                    o.put("items", JsonSchema.getDefaultSchemaNode());
                } else {
                    JsonSerializer<Object> ser = provider.findValueSerializer(componentType, _property);
                    JsonNode schemaNode = (ser instanceof SchemaAware) ?
                            ((SchemaAware) ser).getSchema(provider, null) :
                            JsonSchema.getDefaultSchemaNode();
                    o.put("items", schemaNode);
                }
            }
        }
        return o;
    }

    /**
     * Need to get callback to resolve value serializer, if static typing
     * is used (either being forced, or because value type is final)
     */
    public void resolve(SerializerProvider provider)
        throws JsonMappingException
    {
        if (_staticTyping) {
            _elementSerializer = provider.findValueSerializer(_elementType, _property);
        }
    }

    /**
```

```
     * @since 1.7
     */
    protected final JsonSerializer<Object> _findAndAddDynamic(PropertySerializerMap map,
            Class<?> type, SerializerProvider provider) throws JsonMappingException
    {
        PropertySerializerMap.SerializerAndMapResult result = map.findAndAddSerializer(type, provider, _property);
        // did we get a new map of serializers? If so, start using it
        if (map != result.map) {
            _dynamicSerializers = result.map;
        }
        return result.serializer;
    }

    /**
     * @since 1.8
     */
    protected final JsonSerializer<Object> _findAndAddDynamic(PropertySerializerMap map,
            JavaType type, SerializerProvider provider) throws JsonMappingException
    {
        PropertySerializerMap.SerializerAndMapResult result = map.findAndAddSerializer(type, provider, _property);
        // did we get a new map of serializers? If so, start using it
        if (map != result.map) {
            _dynamicSerializers = result.map;
        }
        return result.serializer;
    }

}
package org.codehaus.jackson.map.ser.impl;

import java.io.IOException;
import java.util.TimeZone;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.ser.ScalarSerializerBase;

/**
* @since 1.8
*/
public class TimeZoneSerializer
    extends ScalarSerializerBase<TimeZone>
{
    public final static TimeZoneSerializer instance = new TimeZoneSerializer();

    public TimeZoneSerializer() { super(TimeZone.class); }
```

```
  @Override
  public void serialize(TimeZone value, JsonGenerator jgen, SerializerProvider provider)
     throws IOException, JsonGenerationException
  {
     jgen.writeString(value.getID());
  }
}
package org.codehaus.jackson.map.ser.impl;

import java.util.Map;

import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.ser.SerializerCache.TypeKey;

/**
* Specialized read-only map used for storing and accessing serializers by type.
*
* @since 1.7
*/
public class JsonSerializerMap
{
  private final Bucket[] _buckets;

  private final int _size;

  public JsonSerializerMap(Map<TypeKey,JsonSerializer<Object>> serializers)
  {
     int size = findSize(serializers.size());
     _size = size;
     int hashMask = (size-1);
     Bucket[] buckets = new Bucket[size];
     for (Map.Entry<TypeKey,JsonSerializer<Object>> entry : serializers.entrySet()) {
        TypeKey key = entry.getKey();
        int index = key.hashCode() & hashMask;
        buckets[index] = new Bucket(buckets[index], key, entry.getValue());
     }
     _buckets = buckets;
  }

  private final static int findSize(int size)
  {
     // For small enough results (64 or less), we'll require <= 50% fill rate; otherwise 80%
     int needed = (size <= 64) ? (size + size) : (size + (size >> 2));
     int result = 8;
     while (result < needed) {
        result += result;
     }
     return result;
```

```
}

/*
/**********************************************************
/* Public API
/**********************************************************
 */

public int size() { return _size; }

public JsonSerializer<Object> find(TypeKey key)
{
   int index = key.hashCode() & (_buckets.length-1);
   Bucket bucket = _buckets[index];
   /* Ok let's actually try unrolling loop slightly as this shows up in profiler;
    * and also because in vast majority of cases first entry is either null
    * or matches.
    */
   if (bucket == null) {
      return null;
   }
   if (key.equals(bucket.key)) {
      return bucket.value;
   }
   while ((bucket = bucket.next) != null) {
      if (key.equals(bucket.key)) {
         return bucket.value;
      }
   }
   return null;
}

/*
/**********************************************************
/* Helper beans
/**********************************************************
 */

private final static class Bucket
{
   public final TypeKey key;
   public final JsonSerializer<Object> value;
   public final Bucket next;

   public Bucket(Bucket next, TypeKey key, JsonSerializer<Object> value)
   {
      this.next = next;
      this.key = key;
```

```
            this.value = value;
        }
    }
}
package org.codehaus.jackson.map.ser.impl;

import java.util.*;

import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.ser.SerializerCache.TypeKey;

/**
* Optimized lookup table for accessing two types of serializers; typed
* and non-typed. Only accessed from a single thread, so no synchronization
* needed for accessors.
*
* @since 1.7
*/
public final class ReadOnlyClassToSerializerMap
{
  /**
   * Actual mappings from type key to serializers
   */
  protected final JsonSerializerMap _map;

  /**
   * We'll reuse key class to avoid unnecessary instantiations; since
   * this is not shared between threads, we can just reuse single
   * instance.
   */
  protected final TypeKey _cacheKey = new TypeKey(getClass(), false);

  private ReadOnlyClassToSerializerMap(JsonSerializerMap map)
  {
     _map = map;
  }

  public ReadOnlyClassToSerializerMap instance()
  {
     return new ReadOnlyClassToSerializerMap(_map);
  }

  /**
   * Factory method for creating the "blueprint" lookup map. Such map
   * can not be used as is but just shared: to get an actual usable
   * instance, {@link #instance} has to be called first.
   */
```

```
  public static ReadOnlyClassToSerializerMap from(HashMap<TypeKey, JsonSerializer<Object>> src)
  {
     return new ReadOnlyClassToSerializerMap(new JsonSerializerMap(src));
  }

  public JsonSerializer<Object> typedValueSerializer(JavaType type)
  {
     _cacheKey.resetTyped(type);
     return _map.find(_cacheKey);
  }

  public JsonSerializer<Object> typedValueSerializer(Class<?> cls)
  {
     _cacheKey.resetTyped(cls);
     return _map.find(_cacheKey);
  }

  public JsonSerializer<Object> untypedValueSerializer(Class<?> cls)
  {
     _cacheKey.resetUntyped(cls);
     return _map.find(_cacheKey);
  }

  public JsonSerializer<Object> untypedValueSerializer(JavaType type)
  {
     _cacheKey.resetUntyped(type);
     return _map.find(_cacheKey);
  }
}
package org.codehaus.jackson.map.ser.impl;

import java.io.IOException;
import java.util.*;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.ResolvableSerializer;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.TypeSerializer;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;

/**
* Efficient implement for serializing {@link List}s that contains Strings and are random-accessible.
* The only complexity is due to possibility that serializer for {@link String}
```

```
* may be overridde; because of this, logic is needed to ensure that the default
* serializer is in use to use fastest mode, or if not, to defer to custom
* String serializer.
*
* @since 1.7
*/
@JacksonStdImpl
public final class IndexedStringListSerializer
  extends StaticListSerializerBase<List<String>>
  implements ResolvableSerializer
{
  protected JsonSerializer<String> _serializer;

  public IndexedStringListSerializer(BeanProperty property) {
    super(List.class, property);
  }

  @Override protected JsonNode contentSchema() {
    return createSchemaNode("string", true);
  }

  @SuppressWarnings("unchecked")
  @Override
  public void resolve(SerializerProvider provider) throws JsonMappingException
  {
    JsonSerializer<?> ser = provider.findValueSerializer(String.class, _property);
    if (!isDefaultSerializer(ser)) {
      _serializer = (JsonSerializer<String>) ser;
    }
  }

  @Override
  public void serialize(List<String> value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
  {
    jgen.writeStartArray();
    if (_serializer == null) {
      serializeContents(value, jgen, provider);
    } else {
      serializeUsingCustom(value, jgen, provider);
    }
    jgen.writeEndArray();
  }

  @Override
  public void serializeWithType(List<String> value, JsonGenerator jgen, SerializerProvider provider,
      TypeSerializer typeSer)
    throws IOException, JsonGenerationException
```

```
{
   typeSer.writeTypePrefixForArray(value, jgen);
   if (_serializer == null) {
      serializeContents(value, jgen, provider);
   } else {
      serializeUsingCustom(value, jgen, provider);
   }
   typeSer.writeTypeSuffixForArray(value, jgen);
}

private final void serializeContents(List<String> value, JsonGenerator jgen, SerializerProvider provider)
   throws IOException, JsonGenerationException
{
   int i = 0;
   try {
      final int len = value.size();
      for (; i < len; ++i) {
         String str = value.get(i);
         if (str == null) {
            provider.defaultSerializeNull(jgen);
         } else {
            jgen.writeString(str);
         }
      }
   } catch (Exception e) {
      wrapAndThrow(provider, e, value, i);
   }
}

private final void serializeUsingCustom(List<String> value, JsonGenerator jgen, SerializerProvider provider)
   throws IOException, JsonGenerationException
{
   int i = 0;
   try {
      final int len = value.size();
      final JsonSerializer<String> ser = _serializer;
      for (i = 0; i < len; ++i) {
         String str = value.get(i);
         if (str == null) {
            provider.defaultSerializeNull(jgen);
         } else {
            ser.serialize(str, jgen, provider);
         }
      }
   } catch (Exception e) {
      wrapAndThrow(provider, e, value, i);
   }
}
```

```
}
package org.codehaus.jackson.map.ser.impl;

import java.util.*;

import org.codehaus.jackson.map.ser.*;

/**
* Simple {@link FilterProvider} implementation that just stores
* direct id-to-filter mapping.
*/
public class SimpleFilterProvider extends FilterProvider
{
  /**
   * Mappings from ids to filters.
   */
  protected final Map<String,BeanPropertyFilter> _filtersById;

  /**
   * This is the filter we return in case no mapping was found for
   * given id; default is 'null' (in which case caller typically
   * reports an error), but can be set to an explicit filter.
   */
  protected BeanPropertyFilter _defaultFilter;

  /*
  /**********************************************************
  /* Life-cycle: constructing, configuring
  /**********************************************************
   */

  public SimpleFilterProvider() {
     _filtersById = new HashMap<String,BeanPropertyFilter>();
  }

  /**
   * @param mapping Mapping from id to filter; used as is, no copy is made.
   */
  public SimpleFilterProvider(Map<String,BeanPropertyFilter> mapping) {
     _filtersById = new HashMap<String,BeanPropertyFilter>();
  }

  /**
   * Method for defining filter to return for "unknown" filters; cases
   * where there is no mapping from given id to an explicit filter.
   *
   * @param f Filter to return when no filter is found for given id
   */
```

```
  public SimpleFilterProvider setDefaultFilter(BeanPropertyFilter f)
  {
     _defaultFilter = f;
     return this;
  }

  public SimpleFilterProvider addFilter(String id, BeanPropertyFilter filter) {
     _filtersById.put(id, filter);
     return this;
  }

  public BeanPropertyFilter removeFilter(String id) {
     return _filtersById.remove(id);
  }

  /*
  /**********************************************************
  /* Public lookup API
  /**********************************************************
   */

  @Override
  public BeanPropertyFilter findFilter(Object filterId)
  {
     BeanPropertyFilter f = _filtersById.get(filterId);
     return (f == null) ? _defaultFilter : f;
  }
}
package org.codehaus.jackson.map.ser.impl;

import java.lang.reflect.Type;
import java.util.*;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.ser.SerializerBase;
import org.codehaus.jackson.node.ObjectNode;

/**
* Intermediate base class for Lists, Collections and Arrays
* that contain static (non-dynamic) value types.
*
* @since 1.7
*/
public abstract class StaticListSerializerBase<T extends Collection<?>>
  extends SerializerBase<T>
{
```

```
  /**
   * Property that contains String List to serialize, if known.
   */
  protected final BeanProperty _property;

  protected StaticListSerializerBase(Class<?> cls, BeanProperty property)
  {
     super(cls, false);
     _property = property;
  }

  @Override
  public JsonNode getSchema(SerializerProvider provider, Type typeHint)
  {
     ObjectNode o = createSchemaNode("array", true);
     o.put("items", contentSchema());
     return o;
  }

  /*
  /**********************************************************
  /* Abstract methods for sub-classes to implement
  /**********************************************************
   */

  protected abstract JsonNode contentSchema();
}
package org.codehaus.jackson.map.ser.impl;

import java.lang.reflect.Type;
import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.ser.SerializerBase;

/**
* This is a simple dummy serializer that will just output raw values by calling toString()
* on value to serialize.
*
* @since 1.7
*/
@JacksonStdImpl
public class RawSerializer<T>
  extends SerializerBase<T>
{
  /**
```

```
     * Constructor takes in expected type of values; but since caller
     * typically can not really provide actual type parameter, we will
     * just take wild card and coerce type.
     */
    public RawSerializer(Class<?> cls) {
        super(cls, false);
    }

    @Override
    public void serialize(T value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeRawValue(value.toString());
    }

    @Override
    public void serializeWithType(T value, JsonGenerator jgen, SerializerProvider provider,
            TypeSerializer typeSer)
        throws IOException, JsonProcessingException
    {
        typeSer.writeTypePrefixForScalar(value, jgen);
        serialize(value, jgen, provider);
        typeSer.writeTypeSuffixForScalar(value, jgen);
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        // type not really known, but since it is a JSON string:
        return createSchemaNode("string", true);
    }
}
package org.codehaus.jackson.map.ser.impl;

import java.io.IOException;
import java.util.*;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.ResolvableSerializer;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.TypeSerializer;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
```

```
/**
* Efficient implement for serializing {@link Collection}s that contain Strings.
* The only complexity is due to possibility that serializer for {@link String}
* may be overridde; because of this, logic is needed to ensure that the default
* serializer is in use to use fastest mode, or if not, to defer to custom
* String serializer.
*
* @since 1.7
*/
@JacksonStdImpl

public class StringCollectionSerializer
  extends StaticListSerializerBase<Collection<String>>
  implements ResolvableSerializer
{
  protected JsonSerializer<String> _serializer;

  public StringCollectionSerializer(BeanProperty property) {
     super(Collection.class, property);
  }

  @Override protected JsonNode contentSchema() {
     return createSchemaNode("string", true);
  }

  @SuppressWarnings("unchecked")
  @Override
  public void resolve(SerializerProvider provider) throws JsonMappingException
  {
     JsonSerializer<?> ser = provider.findValueSerializer(String.class, _property);
     if (!isDefaultSerializer(ser)) {
        _serializer = (JsonSerializer<String>) ser;
     }
  }

  @Override
  public void serialize(Collection<String> value, JsonGenerator jgen, SerializerProvider provider)
     throws IOException, JsonGenerationException
  {
     jgen.writeStartArray();
     if (_serializer == null) {
        serializeContents(value, jgen, provider);
     } else {
        serializeUsingCustom(value, jgen, provider);
     }
     jgen.writeEndArray();
  }
```

```
@Override
public void serializeWithType(Collection<String> value, JsonGenerator jgen, SerializerProvider provider,
        TypeSerializer typeSer)
    throws IOException, JsonGenerationException
{
    typeSer.writeTypePrefixForArray(value, jgen);
    if (_serializer == null) {
        serializeContents(value, jgen, provider);
    } else {
        serializeUsingCustom(value, jgen, provider);
    }
    typeSer.writeTypeSuffixForArray(value, jgen);
}

private final void serializeContents(Collection<String> value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
{
    if (_serializer != null) {
        serializeUsingCustom(value, jgen, provider);
        return;
    }
    int i = 0;
    for (String str : value) {
        try {
            if (str == null) {
                provider.defaultSerializeNull(jgen);
            } else {
                jgen.writeString(str);
            }
            ++i;
        } catch (Exception e) {
            wrapAndThrow(provider, e, value, i);
        }
    }
}

private void serializeUsingCustom(Collection<String> value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
{
    final JsonSerializer<String> ser = _serializer;
    int i = 0;
    for (String str : value) {
        try {
            if (str == null) {
                provider.defaultSerializeNull(jgen);
            } else {
                ser.serialize(str, jgen, provider);
            }
```

```
            } catch (Exception e) {
                wrapAndThrow(provider, e, value, i);
            }
        }
    }

}
package org.codehaus.jackson.map.ser.impl;

import java.util.*;

import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.ser.BeanPropertyFilter;
import org.codehaus.jackson.map.ser.BeanPropertyWriter;

/**
* Simple {@link BeanPropertyFilter} implementation that only uses property name
* to determine whether to serialize property as is, or to filter it out.
*
* @since 1.7
*/
public abstract class SimpleBeanPropertyFilter implements BeanPropertyFilter
{
    /*
    /**********************************************************
    /* Life-cycle
    /**********************************************************
     */

    protected SimpleBeanPropertyFilter() { }

    /**
     * Factory method to construct filter that filters out all properties <b>except</b>
     * ones includes in set
     */
    public static SimpleBeanPropertyFilter filterOutAllExcept(Set<String> properties) {
        return new FilterExceptFilter(properties);
    }

    public static SimpleBeanPropertyFilter filterOutAllExcept(String... propertyArray) {
        HashSet<String> properties = new HashSet<String>(propertyArray.length);
        Collections.addAll(properties, propertyArray);
        return new FilterExceptFilter(properties);
    }

    public static SimpleBeanPropertyFilter serializeAllExcept(Set<String> properties) {
        return new SerializeExceptFilter(properties);
```

```
}

public static SimpleBeanPropertyFilter serializeAllExcept(String... propertyArray) {
   HashSet<String> properties = new HashSet<String>(propertyArray.length);
   Collections.addAll(properties, propertyArray);
   return new SerializeExceptFilter(properties);
}

/*
/**********************************************************
/* Sub-classes
/**********************************************************
 */

/**
 * Filter implementation which defaults to filtering out unknown
 * properties and only serializes ones explicitly listed.
 */
public static class FilterExceptFilter
   extends SimpleBeanPropertyFilter
{
   /**
    * Set of property names to serialize.
    */
   protected final Set<String> _propertiesToInclude;

   public FilterExceptFilter(Set<String> properties) {
      _propertiesToInclude = properties;
   }

   @Override
   public void serializeAsField(Object bean, JsonGenerator jgen,
         SerializerProvider provider, BeanPropertyWriter writer)
      throws Exception
   {
      if (_propertiesToInclude.contains(writer.getName())) {
         writer.serializeAsField(bean, jgen, provider);
      }
   }
}

/**
 * Filter implementation which defaults to serializing all
 * properties, except for ones explicitly listed to be filtered out.
 */
public static class SerializeExceptFilter
   extends SimpleBeanPropertyFilter
{
```

```
    /**
     * Set of property names to filter out.
     */
    protected final Set<String> _propertiesToExclude;

    public SerializeExceptFilter(Set<String> properties) {
        _propertiesToExclude = properties;
    }

    @Override
    public void serializeAsField(Object bean, JsonGenerator jgen,
            SerializerProvider provider, BeanPropertyWriter writer)
        throws Exception
    {
        if (!_propertiesToExclude.contains(writer.getName())) {
            writer.serializeAsField(bean, jgen, provider);
        }
    }
  }
}
package org.codehaus.jackson.map.ser.impl;

import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.type.JavaType;

/**
* Helper container used for resolving serializers for dynamic (possibly but not
* necessarily polymorphic) properties: properties whose type is not forced
* to use dynamic (declared) type and that are not final.
* If so, serializer to use can only be established once actual value type is known.
* Since this happens a lot unless static typing is forced (or types are final)
* this implementation is optimized for efficiency.
* Instances are immutable; new instances are created with factory methods: this
* is important to ensure correct multi-threaded access.
*
* @since 1.7
*/
public abstract class PropertySerializerMap
{
  /**
   * Main lookup method. Takes a "raw" type since usage is always from
   * place where parameterization is fixed such that there can not be
   * type-parametric variations.
   */
  public abstract JsonSerializer<Object> serializerFor(Class<?> type);
```

```
/**
 * Method called if initial lookup fails; will both find serializer
 * and construct new map instance if warranted, and return both
 * @throws JsonMappingException
 */
public final SerializerAndMapResult findAndAddSerializer(Class<?> type,
        SerializerProvider provider, BeanProperty property)
    throws JsonMappingException
{
    JsonSerializer<Object> serializer = provider.findValueSerializer(type, property);
    return new SerializerAndMapResult(serializer, newWith(type, serializer));
}

public final SerializerAndMapResult findAndAddSerializer(JavaType type,
        SerializerProvider provider, BeanProperty property)
    throws JsonMappingException
{
    JsonSerializer<Object> serializer = provider.findValueSerializer(type, property);
    return new SerializerAndMapResult(serializer, newWith(type.getRawClass(), serializer));
}

protected abstract PropertySerializerMap newWith(Class<?> type, JsonSerializer<Object> serializer);

public static PropertySerializerMap emptyMap() {
    return Empty.instance;
}

/*
/**********************************************************
/* Helper classes
/**********************************************************
 */

/**
 * Value class used for returning tuple that has both serializer
 * that was retrieved and new map instance
 */
public final static class SerializerAndMapResult
{
    public final JsonSerializer<Object> serializer;
    public final PropertySerializerMap map;

    public SerializerAndMapResult(JsonSerializer<Object> serializer,
            PropertySerializerMap map)
    {
        this.serializer = serializer;
        this.map = map;
```

```
    }
}

/**
 * Trivial container for bundling type + serializer entries.
 */
private final static class TypeAndSerializer
{
    public final Class<?> type;
    public final JsonSerializer<Object> serializer;

    public TypeAndSerializer(Class<?> type, JsonSerializer<Object> serializer) {
        this.type = type;
        this.serializer = serializer;
    }
}

/*
/**********************************************************
/* Implementations
/**********************************************************
 */

/**
 * Bogus instance that contains no serializers; used as the default
 * map with new serializers.
 */
private final static class Empty extends PropertySerializerMap
{
    protected final static Empty instance = new Empty();

    @Override
    public JsonSerializer<Object> serializerFor(Class<?> type) {
        return null; // empty, nothing to find
    }

    @Override
    protected PropertySerializerMap newWith(Class<?> type, JsonSerializer<Object> serializer) {
        return new Single(type, serializer);
    }
}

/**
 * Map that contains a single serializer; although seemingly silly
 * this is probably the most commonly used variant because many
 * theoretically dynamic or polymorphic types just have single
 * actual type.
 */
```

```
private final static class Single extends PropertySerializerMap
{
    private final Class<?> _type;
    private final JsonSerializer<Object> _serializer;

    public Single(Class<?> type, JsonSerializer<Object> serializer) {
        _type = type;
        _serializer = serializer;
    }

    @Override
    public JsonSerializer<Object> serializerFor(Class<?> type)
    {
        if (type == _type) {
            return _serializer;
        }
        return null;
    }

    @Override
    protected PropertySerializerMap newWith(Class<?> type, JsonSerializer<Object> serializer) {
        return new Double(_type, _serializer, type, serializer);
    }
}

private final static class Double extends PropertySerializerMap
{
    private final Class<?> _type1, _type2;
    private final JsonSerializer<Object> _serializer1, _serializer2;

    public Double(Class<?> type1, JsonSerializer<Object> serializer1,
            Class<?> type2, JsonSerializer<Object> serializer2)
    {
        _type1 = type1;
        _serializer1 = serializer1;
        _type2 = type2;
        _serializer2 = serializer2;
    }

    @Override
    public JsonSerializer<Object> serializerFor(Class<?> type)
    {
        if (type == _type1) {
            return _serializer1;
        }
        if (type == _type2) {
            return _serializer2;
        }
```

```
        return null;
    }

    @Override
    protected PropertySerializerMap newWith(Class<?> type, JsonSerializer<Object> serializer) {
        // Ok: let's just create generic one
        TypeAndSerializer[] ts = new TypeAndSerializer[2];
        ts[0] = new TypeAndSerializer(_type1, _serializer1);
        ts[1] = new TypeAndSerializer(_type2, _serializer2);
        return new Multi(ts);
    }
}

private final static class Multi extends PropertySerializerMap
{
    /**
     * Let's limit number of serializers we actually cache; linear
     * lookup won't scale too well beyond smallish number, and if
     * we really want to support larger collections should use
     * a hash map. But it seems unlikely this is a common use
     * case so for now let's just stop building after hard-coded
     * limit. 8 sounds like a reasonable stab for now.
     */
    private final static int MAX_ENTRIES = 8;

    private final TypeAndSerializer[] _entries;

    public Multi(TypeAndSerializer[] entries) {
        _entries = entries;
    }

    @Override
    public JsonSerializer<Object> serializerFor(Class<?> type)
    {
        for (int i = 0, len = _entries.length; i < len; ++i) {
            TypeAndSerializer entry = _entries[i];
            if (entry.type == type) {
                return entry.serializer;
            }
        }
        return null;
    }

    @Override
    protected PropertySerializerMap newWith(Class<?> type, JsonSerializer<Object> serializer)
    {
        int len = _entries.length;
        // Will only grow up to N entries
```

```
            if (len == MAX_ENTRIES) {
                return this;
            }
            // 1.6 has nice resize methods but we are still 1.5
            TypeAndSerializer[] entries = new TypeAndSerializer[len+1];
            System.arraycopy(_entries, 0, entries, 0, len);
            entries[len] = new TypeAndSerializer(type, serializer);
            return new Multi(entries);
        }
    }
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Type;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.node.JsonNodeFactory;
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.schema.SchemaAware;
import org.codehaus.jackson.type.JavaType;

/**
* Base class used by all standard serializers. Provides some convenience
* methods for implementing {@link SchemaAware}
*/
public abstract class SerializerBase<T>
    extends JsonSerializer<T>
    implements SchemaAware
{
    protected final Class<T> _handledType;

    protected SerializerBase(Class<T> t) {
        _handledType = t;
    }

    /**
     * @since 1.7
     */
    @SuppressWarnings("unchecked")
    protected SerializerBase(JavaType type) {
        _handledType = (Class<T>) type.getRawClass();
    }
```

```
/**
 * Alternate constructor that is (alas!) needed to work
 * around kinks of generic type handling
 */
@SuppressWarnings("unchecked")
protected SerializerBase(Class<?> t, boolean dummy) {
    _handledType = (Class<T>) t;
}

@Override
public final Class<T> handledType() { return _handledType; }

@Override
public abstract void serialize(T value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException;

public abstract JsonNode getSchema(SerializerProvider provider, Type typeHint)
    throws JsonMappingException;

protected ObjectNode createObjectNode() {
    return JsonNodeFactory.instance.objectNode();
}

protected ObjectNode createSchemaNode(String type)
{
    ObjectNode schema = createObjectNode();
    schema.put("type", type);
    return schema;
}

protected ObjectNode createSchemaNode(String type, boolean isOptional)
{
    ObjectNode schema = createSchemaNode(type);
    schema.put("optional", isOptional);
    return schema;
}

/**
 * Method that can be called to determine if given serializer is the default
 * serializer Jackson uses; as opposed to a custom serializer installed by
 * a module or calling application. Determination is done using
 * {@link JacksonStdImpl} annotation on serializer class.
 *
 * @since 1.7
 */
protected boolean isDefaultSerializer(JsonSerializer<?> serializer)
{
```

```
    return (serializer != null && serializer.getClass().getAnnotation(JacksonStdImpl.class) != null);
}

/**
 * Method that will modify caught exception (passed in as argument)
 * as necessary to include reference information, and to ensure it
 * is a subtype of {@link IOException}, or an unchecked exception.
 *<p>
 * Rules for wrapping and unwrapping are bit complicated; essentially:
 *<ul>
 * <li>Errors are to be passed as is (if uncovered via unwrapping)
 * <li>"Plain" IOExceptions (ones that are not of type
 *   {@link JsonMappingException} are to be passed as is
 *</ul>
 */
public void wrapAndThrow(SerializerProvider provider,
        Throwable t, Object bean, String fieldName)
    throws IOException
{
    /* 05-Mar-2009, tatu: But one nasty edge is when we get
     *   StackOverflow: usually due to infinite loop. But that
     *   usually gets hidden within an InvocationTargetException...
     */
    while (t instanceof InvocationTargetException && t.getCause() != null) {
        t = t.getCause();
    }
    // Errors and "plain" IOExceptions to be passed as is
    if (t instanceof Error) {
        throw (Error) t;
    }
    // Ditto for IOExceptions... except for mapping exceptions!
    boolean wrap = (provider == null) || provider.isEnabled(SerializationConfig.Feature.WRAP_EXCEPTIONS);
    if (t instanceof IOException) {
        if (!wrap || !(t instanceof JsonMappingException)) {
            throw (IOException) t;
        }
    } else if (!wrap) { // [JACKSON-407] -- allow disabling wrapping for unchecked exceptions
        if (t instanceof RuntimeException) {
            throw (RuntimeException) t;
        }
    }
    // [JACKSON-55] Need to add reference information
    throw JsonMappingException.wrapWithPath(t, bean, fieldName);
}

public void wrapAndThrow(SerializerProvider provider,
        Throwable t, Object bean, int index)
    throws IOException
```

```
    {
        while (t instanceof InvocationTargetException && t.getCause() != null) {
            t = t.getCause();
        }
        // Errors are to be passed as is
        if (t instanceof Error) {
            throw (Error) t;
        }
        // Ditto for IOExceptions... except for mapping exceptions!
        boolean wrap = (provider == null) || provider.isEnabled(SerializationConfig.Feature.WRAP_EXCEPTIONS);
        if (t instanceof IOException) {
            if (!wrap || !(t instanceof JsonMappingException)) {
                throw (IOException) t;
            }
        } else if (!wrap) { // [JACKSON-407] -- allow disabling wrapping for unchecked exceptions
            if (t instanceof RuntimeException) {
                throw (RuntimeException) t;
            }
        }
        // [JACKSON-55] Need to add reference information
        throw JsonMappingException.wrapWithPath(t, bean, index);
    }

    /**
     * @deprecated Use version that takes <code>SerializerProvider</code> instead.
     */
    @Deprecated
    public void wrapAndThrow(Throwable t, Object bean, String fieldName) throws IOException {
        wrapAndThrow(null, t, bean, fieldName);
    }

    /**
     * @deprecated Use version that takes <code>SerializerProvider</code> instead.
     */
    @Deprecated
    public void wrapAndThrow(Throwable t, Object bean, int index) throws IOException {
        wrapAndThrow(null, t, bean, index);
    }
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.Type;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.SerializerProvider;
```

```
import org.codehaus.jackson.map.JsonMappingException;

/**
* Specialized serializer that can be used as the generic key
* serializer, when serializing {@link java.util.Map}s to Json
* Objects.
*/
public final class StdKeySerializer
  extends SerializerBase<Object>
{
  final static StdKeySerializer instace = new StdKeySerializer();

  public StdKeySerializer() { super(Object.class); }

  @Override
  public void serialize(Object value, JsonGenerator jgen, SerializerProvider provider)
     throws IOException, JsonGenerationException
  {
     String keyStr = (value.getClass() == String.class) ?
        ((String) value) : value.toString();
     jgen.writeFieldName(keyStr);
  }

  @Override
  public JsonNode getSchema(SerializerProvider provider, Type typeHint)
     throws JsonMappingException
  {
     return createSchemaNode("string");
  }
}
package org.codehaus.jackson.map.ser;

import java.lang.reflect.Modifier;
import java.util.*;

import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.type.ClassKey;
import org.codehaus.jackson.type.JavaType;

/**
* Serializer factory implementation that allows for configuring
* mapping between types (classes) and serializers to use, by using
* multiple types of overrides. Existing mappings established by
* {@link BeanSerializerFactory} (and its super class,
* {@link BasicSerializerFactory}) are used if no overrides are
* defined.
*<p>
* Unlike base serializer factories ({@link BasicSerializerFactory},
```

```
* {@link BeanSerializerFactory}), this factory is stateful because
* of configuration settings. It is thread-safe, however, as long as
* all configuration as done before using the factory -- a single
* instance can be shared between providers and mappers.
*<p>
* Configurations currently available are:
*<ul>
* <li>Ability to define explicit mappings between classes and interfaces
*  and serializers to use. These can be either specific ones (class must
*  match exactly) or generic ones (any sub-class or class implementing
*  the interface); specific ones have precedence over generic ones (and
*  precedence between generic ones is not defined).
*  </li>
* <li>Ability to define a single generic base serializer for all Enum
*   types (precedence below specific serializer mapping)
*  </li>
*</ul>
*<p>
* Note: as of version 1.7, this class is not as useful as it used to
* be, due to addition of "modules", which allow simpler addition
* of custom serializers and deserializers. In fact, use of module system
* is recommended even when not exposing serializers or deserializers
* as a pluggable library (just using them locally).
*/
public class CustomSerializerFactory
  extends BeanSerializerFactory
{
  /*
  /**********************************************************
  /* Configuration, direct/special mappings
  /**********************************************************
   */

  /**
   * Direct mappings that are only used for exact class type
   * matches, but not for sub-class checks.
   */
  protected HashMap<ClassKey,JsonSerializer<?>> _directClassMappings = null;

  /**
   * And for Enum handling we may specify a single default
   * serializer to use, regardless of actual enumeration.
   * Usually used to provide "toString - serializer".
   */
  protected JsonSerializer<?> _enumSerializerOverride;

  /*
  /**********************************************************
```

```
/* Configuration, generic (interface, super-class) mappings
/**********************************************************
 */

/**
 * And then class-based mappings that are used both for exact and
 * sub-class matches.
 */
protected HashMap<ClassKey,JsonSerializer<?>> _transitiveClassMappings = null;

/**
 * And finally interface-based matches.
 */
protected HashMap<ClassKey,JsonSerializer<?>> _interfaceMappings = null;

/*
/**********************************************************
/* Life-cycle, constructors
/**********************************************************
 */

public CustomSerializerFactory() {
   this(null);
}

public CustomSerializerFactory(Config config) {
   super(config);
}

@Override
public SerializerFactory withConfig(Config config)
{
   /* 22-Nov-2010, tatu: As with BeanSerializerFactory, must ensure type won't change
    *   with this method, so:
    */
   if (getClass() != CustomSerializerFactory.class) {
      throw new IllegalStateException("Subtype of CustomSerializerFactory ("+getClass().getName()
            +") has not properly overridden method 'withAdditionalSerializers': can not instantiate subtype with "
            +"additional serializer definitions");
   }
   return new CustomSerializerFactory(config);
}

/*
/**********************************************************
/* Configuration: type-to-serializer mappings
/**********************************************************
 */
```

```
/**
 * Method used to add a generic (transitive) mapping from specified
 * class or its sub-classes into a serializer.
 * When resolving a type into a serializer, explicit class is checked
 * first, then immediate super-class, and so forth along inheritance
 * chain. But if this fails, implemented interfaces are checked;
 * ordering is done such that first interfaces implemented by
 * the exact type are checked (in order returned by
 * {@link Class#getInterfaces}), then super-type's and so forth.
 *<p>
 * Note that adding generic mappings may lead to problems with
 * sub-classing: if sub-classes add new properties, these may not
 * get properly serialized.
 *
 * @param type Class for which specified serializer is to be
 *   used. May be more specific type than what serializer indicates,
 *   but must be compatible (same or sub-class)
 */
public <T> void addGenericMapping(Class<? extends T> type, JsonSerializer<T> ser)
{
   // Interface to match?
   ClassKey key = new ClassKey(type);
   if (type.isInterface()) {
      if (_interfaceMappings == null) {
         _interfaceMappings = new HashMap<ClassKey,JsonSerializer<?>>();
      }
      _interfaceMappings.put(key, ser);
   } else { // nope, class:
      if (_transitiveClassMappings == null) {
         _transitiveClassMappings = new HashMap<ClassKey,JsonSerializer<?>>();
      }
      _transitiveClassMappings.put(key, ser);
   }
}

/**
 * Method used to add a mapping from specific type -- and only that
 * type -- to specified serializer. This means that binding is not
 * used for sub-types. It also means that no such mappings are to
 * be defined for abstract classes or interfaces: and if an attempt
 * is made, {@link IllegalArgumentException} will be thrown to
 * indicate caller error.
 *
 * @param forClass Class for which specified serializer is to be
 *   used. May be more specific type than what serializer indicates,
 *   but must be compatible (same or sub-class)
 */
```

```
  public <T> void addSpecificMapping(Class<? extends T> forClass, JsonSerializer<T> ser)
  {
     ClassKey key = new ClassKey(forClass);

     /* First, let's ensure it's not an interface or abstract class:
      * as those can not be instantiated, such mappings would never
      * get used.
      */
     if (forClass.isInterface()) {
        throw new IllegalArgumentException("Can not add specific mapping for an interface
("+forClass.getName()+")");
     }
     if (Modifier.isAbstract(forClass.getModifiers())) {
        throw new IllegalArgumentException("Can not add specific mapping for an abstract class
("+forClass.getName()+")");
     }

     if (_directClassMappings == null) {
        _directClassMappings = new HashMap<ClassKey,JsonSerializer<?>>();
     }
     _directClassMappings.put(key, ser);
  }

  /**
   * Method that can be used to force specified serializer to be used for
   * serializing all Enum instances. This is most commonly used to specify
   * serializers that call either <code>enum.toString()</code>, or modify
   * value returned by <code>enum.name()</code> (such as upper- or
   * lower-casing it).
   *<p>
   * Note: this serializer has lower precedence than that of specific
   * types; so if a specific serializer is assigned to an Enum type,
   * this serializer will NOT be used. It has higher precedence than
   * generic mappings have however.
   */
  public void setEnumSerializer(JsonSerializer<?> enumSer)
  {
     _enumSerializerOverride = enumSer;
  }

  /*
  /**********************************************************
  /* JsonSerializerFactory impl
  /**********************************************************
   */

  @Override
  @SuppressWarnings("unchecked")
```

```
public JsonSerializer<Object> createSerializer(SerializationConfig config, JavaType type,
        BeanProperty property)
{
    JsonSerializer<?> ser = findCustomSerializer(type.getRawClass(), config);
    if (ser != null) {
        return (JsonSerializer<Object>) ser;
    }
    return super.createSerializer(config, type, property);
}

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

protected JsonSerializer<?> findCustomSerializer(Class<?> type, SerializationConfig config)
{
    JsonSerializer<?> ser = null;
    ClassKey key = new ClassKey(type);

    // First: exact matches
    if (_directClassMappings != null) {
        ser = _directClassMappings.get(key);
        if (ser != null) {
            return ser;
        }
    }

    // No match? Perhaps we can use the enum serializer?
    if (type.isEnum()) {
        if (_enumSerializerOverride != null) {
            return _enumSerializerOverride;
        }
    }

    // Still no match? How about more generic ones?
    // Mappings for super-classes?
    if (_transitiveClassMappings != null) {
        for (Class<?> curr = type; (curr != null); curr = curr.getSuperclass()) {
            key.reset(curr);
            ser = _transitiveClassMappings.get(key);
            if (ser != null) {
                return ser;
            }
        }
    }
```

```
        // And if still no match, how about interfaces?
        if (_interfaceMappings != null) {
            // as per [JACKSON-327], better check actual interface first too...
            key.reset(type);
            ser = _interfaceMappings.get(key);
            if (ser != null) {
                return ser;
            }
            for (Class<?> curr = type; (curr != null); curr = curr.getSuperclass()) {
                ser = _findInterfaceMapping(curr, key);
                if (ser != null) {
                    return ser;
                }
            }
        }
        return null;
    }

    protected JsonSerializer<?> _findInterfaceMapping(Class<?> cls, ClassKey key)
    {
        for (Class<?> iface : cls.getInterfaces()) {
            key.reset(iface);
            JsonSerializer<?> ser = _interfaceMappings.get(key);
            if (ser != null) {
                return ser;
            }
            // [JACKSON-373] need to also check super-interfaces
            ser = _findInterfaceMapping(iface, key);
            if (ser != null) {
                return ser;
            }
        }
        return null;
    }
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.Type;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Calendar;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
```

```
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.schema.JsonSerializableSchema;
import org.codehaus.jackson.util.TokenBuffer;

/**
* Container class for serializers used for handling standard JDK-provided
* types
*
* @since 1.5
*/
public class StdSerializers
{
  protected StdSerializers() { }

  /*
  /**********************************************************
  /* Abstract base classes
  /**********************************************************
   */

  /**
   * Intermediate base class for limited number of scalar types
   * that should never include type information. These are "native"
   * types that are default mappings for corresponding JSON scalar
   * types: String, Integer, Double and Boolean.
   */
  protected abstract static class NonTypedScalarSerializer<T>
     extends ScalarSerializerBase<T>
  {
     protected NonTypedScalarSerializer(Class<T> t) {
        super(t);
     }

     @Override
     public final void serializeWithType(T value, JsonGenerator jgen, SerializerProvider provider,
           TypeSerializer typeSer)
        throws IOException, JsonGenerationException
     {
        // no type info, just regular serialization
        serialize(value, jgen, provider);
     }
  }

  /*
  /**********************************************************
  /* Concrete serializers, non-numeric primitives, Strings, Classes
  /**********************************************************
```

```
 */

/**
 * Serializer used for primitive boolean, as well as java.util.Boolean
 * wrapper type.
 *<p>
 * Since this is one of "native" types, no type information is ever
 * included on serialization (unlike for most scalar types as of 1.5)
 */
@JacksonStdImpl
public final static class BooleanSerializer
   extends NonTypedScalarSerializer<Boolean>
{
   /**
    * Whether type serialized is primitive (boolean) or wrapper
    * (java.lang.Boolean); if true, former, if false, latter.
    */
   final boolean _forPrimitive;

   public BooleanSerializer(boolean forPrimitive)
   {
      super(Boolean.class);
      _forPrimitive = forPrimitive;
   }

   @Override
   public void serialize(Boolean value, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonGenerationException
   {
      jgen.writeBoolean(value.booleanValue());
   }

   @Override
   public JsonNode getSchema(SerializerProvider provider, Type typeHint)
   {
      /*(ryan) it may not, in fact, be optional, but there's no way
       * to tell whether we're referencing a boolean or java.lang.Boolean.
       */
      /* 27-Jun-2009, tatu: Now we can tell, after passing
       *   'forPrimitive' flag...
       */
      return createSchemaNode("boolean", !_forPrimitive);
   }
}

/**
 * This is the special serializer for regular {@link java.lang.String}s.
 *<p>
```

```
 * Since this is one of "native" types, no type information is ever
 * included on serialization (unlike for most scalar types as of 1.5)
 */
@JacksonStdImpl
public final static class StringSerializer
   extends NonTypedScalarSerializer<String>
{
   public StringSerializer() { super(String.class); }

   @Override
   public void serialize(String value, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonGenerationException
   {
      jgen.writeString(value);
   }

   @Override
   public JsonNode getSchema(SerializerProvider provider, Type typeHint)
   {
      return createSchemaNode("string", true);
   }
}

/*
/**********************************************************
/* Concrete serializers, numerics
/**********************************************************
 */

/**
 * This is the special serializer for regular {@link java.lang.Integer}s
 * (and primitive ints)
 *<p>
 * Since this is one of "native" types, no type information is ever
 * included on serialization (unlike for most scalar types as of 1.5)
 */
@JacksonStdImpl
public final static class IntegerSerializer
   extends NonTypedScalarSerializer<Integer>
{
   public IntegerSerializer() { super(Integer.class); }

   @Override
   public void serialize(Integer value, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonGenerationException
   {
      jgen.writeNumber(value.intValue());
   }
```

```
    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        return createSchemaNode("integer", true);
    }
}

/**
 * Similar to {@link IntegerSerializer}, but will not cast to Integer:
 * instead, cast is to {@link java.lang.Number}, and conversion is
 * by calling {@link java.lang.Number#intValue}.
 */
@JacksonStdImpl
public final static class IntLikeSerializer
    extends ScalarSerializerBase<Number>
{
    final static IntLikeSerializer instance = new IntLikeSerializer();

    public IntLikeSerializer() { super(Number.class); }

    @Override
    public void serialize(Number value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeNumber(value.intValue());
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        return createSchemaNode("integer", true);
    }
}

@JacksonStdImpl
public final static class LongSerializer
    extends ScalarSerializerBase<Long>
{
    final static LongSerializer instance = new LongSerializer();

    public LongSerializer() { super(Long.class); }

    @Override
    public void serialize(Long value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeNumber(value.longValue());
```

```
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        return createSchemaNode("number", true);
    }
}

@JacksonStdImpl
public final static class FloatSerializer
    extends ScalarSerializerBase<Float>
{
    final static FloatSerializer instance = new FloatSerializer();

    public FloatSerializer() { super(Float.class); }

    @Override
    public void serialize(Float value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeNumber(value.floatValue());
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        return createSchemaNode("number", true);
    }
}

/**
 * This is the special serializer for regular {@link java.lang.Double}s
 * (and primitive doubles)
 *<p>
 * Since this is one of "native" types, no type information is ever
 * included on serialization (unlike for most scalar types as of 1.5)
 */
@JacksonStdImpl
public final static class DoubleSerializer
    extends NonTypedScalarSerializer<Double>
{
    final static DoubleSerializer instance = new DoubleSerializer();

    public DoubleSerializer() { super(Double.class); }

    @Override
    public void serialize(Double value, JsonGenerator jgen, SerializerProvider provider)
```

```
        throws IOException, JsonGenerationException
    {
        jgen.writeNumber(value.doubleValue());
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        return createSchemaNode("number", true);
    }
}

/**
 * As a fallback, we may need to use this serializer for other
 * types of {@link Number}s (custom types).
 */
@JacksonStdImpl
public final static class NumberSerializer
    extends ScalarSerializerBase<Number>
{
    public final static NumberSerializer instance = new NumberSerializer();

    public NumberSerializer() { super(Number.class); }

    @Override
    public void serialize(Number value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        // As per [JACKSON-423], handling for BigInteger and BigDecimal was missing!
        if (value instanceof BigDecimal) {
            jgen.writeNumber((BigDecimal) value);
        } else if (value instanceof BigInteger) {
            jgen.writeNumber((BigInteger) value);

        /* These shouldn't match (as there are more specific ones),
         * but just to be sure:
         */
        } else if (value instanceof Double) {
            jgen.writeNumber(((Double) value).doubleValue());
        } else if (value instanceof Float) {
            jgen.writeNumber(((Float) value).floatValue());
        } else {
            // We'll have to use fallback "untyped" number write method
            jgen.writeNumber(value.toString());
        }
    }

    @Override
```

```
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        return createSchemaNode("number", true);
    }
}

/*
/**********************************************************
/* Serializers for JDK date/time data types
/**********************************************************
 */

/**
 * For time values we should use timestamp, since that is about the only
 * thing that can be reliably converted between date-based objects
 * and json.
 */
@JacksonStdImpl
public final static class CalendarSerializer
    extends ScalarSerializerBase<Calendar>
{
    public final static CalendarSerializer instance = new CalendarSerializer();

    public CalendarSerializer() { super(Calendar.class); }

    @Override
    public void serialize(Calendar value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        provider.defaultSerializeDateValue(value.getTimeInMillis(), jgen);
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        //TODO: (ryan) add a format for the date in the schema?
        return
createSchemaNode(provider.isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)
                ? "number" : "string", true);
    }
}

/**
 * For efficiency, we will serialize Dates as longs, instead of
 * potentially more readable Strings.
 */
@JacksonStdImpl
public final static class UtilDateSerializer
```

```
    extends ScalarSerializerBase<java.util.Date>
{
    public final static UtilDateSerializer instance = new UtilDateSerializer();

    public UtilDateSerializer() { super(java.util.Date.class); }

    @Override
    public void serialize(java.util.Date value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        provider.defaultSerializeDateValue(value, jgen);
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        //todo: (ryan) add a format for the date in the schema?
        return
createSchemaNode(provider.isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)
                ? "number" : "string", true);
    }
}

/**
 * Compared to regular {@link UtilDateSerializer}, we do use String
 * representation here. Why? Basically to truncate of time part, since
 * that should not be used by plain SQL date.
 */
@JacksonStdImpl
public final static class SqlDateSerializer
    extends ScalarSerializerBase<java.sql.Date>
{
    public SqlDateSerializer() { super(java.sql.Date.class); }

    @Override
    public void serialize(java.sql.Date value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeString(value.toString());
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        //todo: (ryan) add a format for the date in the schema?
        return createSchemaNode("string", true);
    }
}
```

```
@JacksonStdImpl
public final static class SqlTimeSerializer
   extends ScalarSerializerBase<java.sql.Time>
{
   public SqlTimeSerializer() { super(java.sql.Time.class); }

   @Override
   public void serialize(java.sql.Time value, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonGenerationException
   {
      jgen.writeString(value.toString());
   }

   @Override
   public JsonNode getSchema(SerializerProvider provider, Type typeHint)
   {
      return createSchemaNode("string", true);
   }
}


/*
/**********************************************************
/* Other serializers
/**********************************************************
 */

/**
 * Generic handler for types that implement {@link JsonSerializable}.
 *<p>
 * Note: given that this is used for anything that implements
 * interface, can not be checked for direct class equivalence.
 */
@JacksonStdImpl
@SuppressWarnings("deprecation")
public final static class SerializableSerializer
   extends SerializerBase<JsonSerializable>
{
   protected final static SerializableSerializer instance = new SerializableSerializer();

   private SerializableSerializer() { super(JsonSerializable.class); }

   @Override
   public void serialize(JsonSerializable value, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonGenerationException
   {
      value.serialize(jgen, provider);
```

```
}

@Override
public final void serializeWithType(JsonSerializable value, JsonGenerator jgen, SerializerProvider provider,
        TypeSerializer typeSer)
    throws IOException, JsonGenerationException
{
    /* 24-Jan-2009, tatus: This is not quite optimal (perhaps we should
     *   just create separate serializer...), but works until 2.0 will
     *   deprecate non-typed interface
     */
    if (value instanceof JsonSerializableWithType) {
        ((JsonSerializableWithType) value).serializeWithType(jgen, provider, typeSer);
    } else {
        this.serialize(value, jgen, provider);
    }
}

@Override
public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    throws JsonMappingException
{
    ObjectNode objectNode = createObjectNode();
    String schemaType = "any";
    String objectProperties = null;
    String itemDefinition = null;
    if (typeHint != null) {
        Class<?> rawClass = TypeFactory.type(typeHint).getRawClass();
        if (rawClass.isAnnotationPresent(JsonSerializableSchema.class)) {
            JsonSerializableSchema schemaInfo = rawClass.getAnnotation(JsonSerializableSchema.class);
            schemaType = schemaInfo.schemaType();
            if (!"##irrelevant".equals(schemaInfo.schemaObjectPropertiesDefinition())) {
                objectProperties = schemaInfo.schemaObjectPropertiesDefinition();
            }
            if (!"##irrelevant".equals(schemaInfo.schemaItemDefinition())) {
                itemDefinition = schemaInfo.schemaItemDefinition();
            }
        }
    }
    objectNode.put("type", schemaType);
    if (objectProperties != null) {
        try {
            objectNode.put("properties", new ObjectMapper().readValue(objectProperties, JsonNode.class));
        } catch (IOException e) {
            throw new IllegalStateException(e);
        }
    }
    if (itemDefinition != null) {
```

```
            try {
                objectNode.put("items", new ObjectMapper().readValue(itemDefinition, JsonNode.class));
            } catch (IOException e) {
                throw new IllegalStateException(e);
            }
        }
        objectNode.put("optional", true);
        return objectNode;
    }
}

/**
 * Generic handler for types that implement {@link JsonSerializableWithType}.
 *<p>
 * Note: given that this is used for anything that implements
 * interface, can not be checked for direct class equivalence.
 */
@JacksonStdImpl
public final static class SerializableWithTypeSerializer
    extends SerializerBase<JsonSerializableWithType>
{
    protected final static SerializableWithTypeSerializer instance = new SerializableWithTypeSerializer();

    private SerializableWithTypeSerializer() { super(JsonSerializableWithType.class); }

    @SuppressWarnings("deprecation") // why is this needed?
    @Override
    public void serialize(JsonSerializableWithType value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        value.serialize(jgen, provider);
    }

    @Override
    public final void serializeWithType(JsonSerializableWithType value, JsonGenerator jgen, SerializerProvider
provider,
            TypeSerializer typeSer)
        throws IOException, JsonGenerationException
    {
        value.serializeWithType(jgen, provider, typeSer);
    }

    // copied verbatim from "JsonSerializableSerializer"
    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
        throws JsonMappingException
    {
        ObjectNode objectNode = createObjectNode();
```

```
        String schemaType = "any";
        String objectProperties = null;
        String itemDefinition = null;
        if (typeHint != null) {
            Class<?> rawClass = TypeFactory.type(typeHint).getRawClass();
            if (rawClass.isAnnotationPresent(JsonSerializableSchema.class)) {
                JsonSerializableSchema schemaInfo = rawClass.getAnnotation(JsonSerializableSchema.class);
                schemaType = schemaInfo.schemaType();
                if (!"##irrelevant".equals(schemaInfo.schemaObjectPropertiesDefinition())) {
                    objectProperties = schemaInfo.schemaObjectPropertiesDefinition();
                }
                if (!"##irrelevant".equals(schemaInfo.schemaItemDefinition())) {
                    itemDefinition = schemaInfo.schemaItemDefinition();
                }
            }
        }
        objectNode.put("type", schemaType);
        if (objectProperties != null) {
            try {
                objectNode.put("properties", new ObjectMapper().readValue(objectProperties, JsonNode.class));
            } catch (IOException e) {
                throw new IllegalStateException(e);
            }
        }
        if (itemDefinition != null) {
            try {
                objectNode.put("items", new ObjectMapper().readValue(itemDefinition, JsonNode.class));
            } catch (IOException e) {
                throw new IllegalStateException(e);
            }
        }
        objectNode.put("optional", true);
        return objectNode;
    }
}

/**
 * We also want to directly support serialization of {@link TokenBuffer};
 * and since it is part of core package, it can not implement
 * {@link JsonSerializable} (which is only included in the mapper
 * package)
 *
 * @since 1.5
 */
@JacksonStdImpl
public final static class TokenBufferSerializer
    extends SerializerBase<TokenBuffer>
{
```

```
    public TokenBufferSerializer() { super(TokenBuffer.class); }

    @Override
    public void serialize(TokenBuffer value, JsonGenerator jgen, SerializerProvider provider)
       throws IOException, JsonGenerationException
    {
       value.serialize(jgen);
    }

    /**
     * Implementing typed output for contents of a TokenBuffer is very tricky,
     * since we do not know for sure what its contents might look like (or, rather,
     * we do know when serializing, but not necessarily when deserializing!)
     * One possibility would be to check the current token, and use that to
     * determine if we would output JSON Array, Object or scalar value.
     * Jackson 1.5 did NOT include any type information; but this seems wrong,
     * and so 1.6 WILL include type information.
     *<p>
     * Note that we just claim it is scalar; this should work ok and is simpler
     * than doing introspection on both serialization and deserialization.
     */
    @Override
    public final void serializeWithType(TokenBuffer value, JsonGenerator jgen, SerializerProvider provider,
           TypeSerializer typeSer)
       throws IOException, JsonGenerationException
    {
       typeSer.writeTypePrefixForScalar(value, jgen);
       serialize(value, jgen, provider);
       typeSer.writeTypeSuffixForScalar(value, jgen);
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
       /* 01-Jan-2010, tatu: Not 100% sure what we should say here:
        *   type is basically not known. This seems closest
        *   approximation
        */
       return createSchemaNode("any", true);
    }
  }
}
package org.codehaus.jackson.map.ser;

import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.JsonSerializer;
```

```
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.TypeSerializer;
import org.codehaus.jackson.map.introspect.AnnotatedMember;
import org.codehaus.jackson.map.ser.impl.PropertySerializerMap;
import org.codehaus.jackson.map.util.Annotations;
import org.codehaus.jackson.io.SerializedString;
import org.codehaus.jackson.type.JavaType;

import java.lang.annotation.Annotation;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.lang.reflect.Type;
import java.util.HashMap;

/**
* Base bean property handler class, which implements common parts of
* reflection-based functionality for accessing a property value
* and serializing it.
*/
public class BeanPropertyWriter
  implements BeanProperty
{
  /*
  /**********************************************************
  /* Settings for accessing property value to serialize
  /**********************************************************
   */

  /**
   * Member (field, method) that represents property and allows access
   * to associated annotations.
   *
   * @since 1.7
   */
  protected final AnnotatedMember _member;

  /**
   * Annotations from context (most often, class that declares property,
   * or in case of sub-class serializer, from that sub-class)
   */
  protected final Annotations _contextAnnotations;

  /**
   * Type property is declared to have, either in class definition
   * or associated annotations.
   */
  protected final JavaType _declaredType;
```

```
/**
 * Accessor method used to get property value, for
 * method-accessible properties.
 * Null if and only if {@link #_field} is null.
 */
protected final Method _accessorMethod;

/**
 * Field that contains the property value for field-accessible
 * properties.
 * Null if and only if {@link #_accessorMethod} is null.
 */
protected final Field _field;

/*
/**********************************************************
/* Opaque internal data that bean serializer factory and
/* bean serializers can add.
/*
/* @since 1.7
/**********************************************************
 */

protected HashMap<Object,Object> _internalSettings;

/*
/**********************************************************
/* Serialization settings
/**********************************************************
 */

/**
 * Logical name of the property; will be used as the field name
 * under which value for the property is written.
 */
protected final SerializedString _name;

/**
 * Type to use for locating serializer; normally same as return
 * type of the accessor method, but may be overridden by annotations.
 */
protected final JavaType _cfgSerializationType;

/**
 * Serializer to use for writing out the value: null if it can not
 * be known statically; non-null if it can.
 */
protected final JsonSerializer<Object> _serializer;
```

```
/**
 * In case serializer is not known statically (i.e. <code>_serializer</code>
 * is null), we will use a lookup structure for storing dynamically
 * resolved mapping from type(s) to serializer(s).
 *
 * @since 1.7
 */
protected PropertySerializerMap _dynamicSerializers;

/**
 * Flag to indicate that null values for this property are not
 * to be written out. That is, if property has value null,
 * no entry will be written
 */
protected final boolean _suppressNulls;

/**
 * Value that is considered default value of the property; used for
 * default-value-suppression if enabled.
 */
protected final Object _suppressableValue;

/**
 * Alternate set of property writers used when view-based filtering
 * is available for the Bean.
 *
 * @since 1.4
 */
protected Class<?>[] _includeInViews;

/**
 * If property being serialized needs type information to be
 * included this is the type serializer to use.
 * Declared type (possibly augmented with annotations) of property
 * is used for determining exact mechanism to use (compared to
 * actual runtime type used for serializing actual state).
 */
protected TypeSerializer _typeSerializer;

/**
 * Base type of the property, if the declared type is "non-trivial";
 * meaning it is either a structured type (collection, map, array),
 * or parametrized. Used to retain type information about contained
 * type, which is mostly necessary if type metadata is to be
 * included.
 *
 * @since 1.5
```

```
 */
protected JavaType _nonTrivialBaseType;

/*
/**********************************************************
/* Construction, configuration
/**********************************************************
 */

public BeanPropertyWriter(AnnotatedMember member, Annotations contextAnnotations,
        String name, JavaType declaredType,
        JsonSerializer<Object> ser, TypeSerializer typeSer, JavaType serType,
        Method m, Field f,
        boolean suppressNulls, Object suppressableValue)
{
    this(member, contextAnnotations, new SerializedString(name), declaredType,
            ser, typeSer, serType,
            m, f, suppressNulls, suppressableValue);
}

public BeanPropertyWriter(AnnotatedMember member, Annotations contextAnnotations,
        SerializedString name, JavaType declaredType,
        JsonSerializer<Object> ser, TypeSerializer typeSer, JavaType serType,
        Method m, Field f, boolean suppressNulls, Object suppressableValue)
{
    _member = member;
    _contextAnnotations = contextAnnotations;
    _name = name;
    _declaredType = declaredType;
    _serializer = ser;
    _dynamicSerializers = (ser == null) ? PropertySerializerMap.emptyMap() : null;
    _typeSerializer = typeSer;
    _cfgSerializationType = serType;
    _accessorMethod = m;
    _field = f;
    _suppressNulls = suppressNulls;
    _suppressableValue = suppressableValue;
}

/**
 * "Copy constructor" to be used by filtering sub-classes
 */
protected BeanPropertyWriter(BeanPropertyWriter base)
{
    this(base, base._serializer);
}

/**
```

```
     * "Copy constructor" to be used by filtering sub-classes
     */
    protected BeanPropertyWriter(BeanPropertyWriter base, JsonSerializer<Object> ser)
    {
        _serializer = ser;

        _member = base._member;
        _contextAnnotations = base._contextAnnotations;
        _name = base._name;
        _declaredType = base._declaredType;
        _dynamicSerializers = base._dynamicSerializers;
        _typeSerializer = base._typeSerializer;
        _cfgSerializationType = base._cfgSerializationType;
        _accessorMethod = base._accessorMethod;
        _field = base._field;
        _suppressNulls = base._suppressNulls;
        _suppressableValue = base._suppressableValue;
        // one more thing: copy internal settings, if any (since 1.7)
        if (base._internalSettings != null) {
            _internalSettings = new HashMap<Object,Object>(base._internalSettings);
        }
    }

    /**
     * Method that will construct and return a new writer that has
     * same properties as this writer, but uses specified serializer
     * instead of currently configured one (if any).
     */
    public BeanPropertyWriter withSerializer(JsonSerializer<Object> ser)
    {
        // sanity check to ensure sub-classes override...
        if (getClass() != BeanPropertyWriter.class) {
            throw new IllegalStateException("BeanPropertyWriter sub-class does not override 'withSerializer()'; needs
to!");
        }
        return new BeanPropertyWriter(this, ser);
    }

    /**
     * Method for defining which views to included value of this
     * property in. If left undefined, will always be included;
     * otherwise active view definition will be checked against
     * definition list and value is only included if active
     * view is one of defined views, or its sub-view (as defined
     * by class/sub-class relationship).
     */
    public void setViews(Class<?>[] views) { _includeInViews = views; }
```

```
/**
 * Method called to define type to consider as "non-trivial" basetype,
 * needed for dynamic serialization resolution for complex (usually container)
 * types
 *
 * @since 1.5
 */
public void setNonTrivialBaseType(JavaType t) {
   _nonTrivialBaseType = t;
}

/*
/**********************************************************
/* BeanProperty impl
/**********************************************************
 */

public String getName() {
   return _name.getValue();
}

public JavaType getType() {
   return _declaredType;
}

public <A extends Annotation> A getAnnotation(Class<A> acls) {
   return _member.getAnnotation(acls);
}

public <A extends Annotation> A getContextAnnotation(Class<A> acls) {
   return _contextAnnotations.get(acls);
}

public AnnotatedMember getMember() {
   return _member;
}

/*
/**********************************************************
/* Managing and accessing of opaque internal settings
/* (used by extensions)
/**********************************************************
 */

/**
 * Method for accessing value of specified internal setting.
 *
 * @return Value of the setting, if any; null if none.
```

```
 *
 * @since 1.7
 */
public Object getInternalSetting(Object key)
{
   if (_internalSettings == null) {
      return null;
   }
   return _internalSettings.get(key);
}

/**
 * Method for setting specific internal setting to given value
 *
 * @return Old value of the setting, if any (null if none)
 *
 * @since 1.7
 */
public Object setInternalSetting(Object key, Object value)
{
   if (_internalSettings == null) {
      _internalSettings = new HashMap<Object,Object>();
   }
   return _internalSettings.put(key, value);
}

/**
 * Method for removing entry for specified internal setting.
 *
 * @return Existing value of the setting, if any (null if none)
 *
 * @since 1.7
 */
public Object removeInternalSetting(Object key)
{
   Object removed = null;
   if (_internalSettings != null) {
      removed = _internalSettings.remove(key);
      // to reduce memory usage, let's also drop the Map itself, if empty
      if (_internalSettings.size() == 0) {
         _internalSettings = null;
      }
   }
   return removed;
}

/*
/**********************************************************
```

```
/* Accessors
/**********************************************************
 */

public SerializedString getSerializedName() { return _name; }

public boolean hasSerializer() { return _serializer != null; }

// Needed by BeanSerializer#getSchema
protected JsonSerializer<Object> getSerializer() {
   return _serializer;
}

public JavaType getSerializationType() {
   return _cfgSerializationType;
}

public Class<?> getRawSerializationType() {
   return (_cfgSerializationType == null) ? null : _cfgSerializationType.getRawClass();
}

public Class<?> getPropertyType()
{
   if (_accessorMethod != null) {
      return _accessorMethod.getReturnType();
   }
   return _field.getType();
}

/**
 * Get the generic property type of this property writer.
 *
 * @return The property type, or null if not found.
 */
public Type getGenericPropertyType()
{
   if (_accessorMethod != null) {
      return _accessorMethod.getGenericReturnType();
   }
   return _field.getGenericType();
}

public Class<?>[] getViews() { return _includeInViews; }

/*
/**********************************************************
/* Serialization functionality
/**********************************************************
```

```
 */

/**
 * Method called to access property that this bean stands for, from
 * within given bean, and to serialize it as a JSON Object field
 * using appropriate serializer.
 */
public void serializeAsField(Object bean, JsonGenerator jgen, SerializerProvider prov)
   throws Exception
{
   Object value = get(bean);
   // Null handling is bit different, check that first
   if (value == null) {
      if (!_suppressNulls) {
         jgen.writeFieldName(_name);
         prov.defaultSerializeNull(jgen);
      }
      return;
   }
   // For non-nulls, first: simple check for direct cycles
   if (value == bean) {
      _reportSelfReference(bean);
   }
   if (_suppressableValue != null && _suppressableValue.equals(value)) {
      return;
   }

   JsonSerializer<Object> ser = _serializer;
   if (ser == null) {
      Class<?> cls = value.getClass();
      PropertySerializerMap map = _dynamicSerializers;
      ser = map.serializerFor(cls);
      if (ser == null) {
         ser = _findAndAddDynamic(map, cls, prov);
      }
   }
   jgen.writeFieldName(_name);
   if (_typeSerializer == null) {
      ser.serialize(value, jgen, prov);
   } else {
      ser.serializeWithType(value, jgen, prov, _typeSerializer);
   }
}

/**
 * @since 1.7
 */
protected final JsonSerializer<Object> _findAndAddDynamic(PropertySerializerMap map,
```

```
        Class<?> type, SerializerProvider provider) throws JsonMappingException
    {
        PropertySerializerMap.SerializerAndMapResult result;
        if (_nonTrivialBaseType != null) {
            JavaType t = _nonTrivialBaseType.forcedNarrowBy(type);
            result = map.findAndAddSerializer(t, provider, this);
        } else {
            result = map.findAndAddSerializer(type, provider, this);
        }
        // did we get a new map of serializers? If so, start using it
        if (map != result.map) {
            _dynamicSerializers = result.map;
        }
        return result.serializer;
    }

    /**
     * Method that can be used to access value of the property this
     * Object describes, from given bean instance.
     *<p>
     * Note: method is final as it should not need to be overridden -- rather,
     * calling method(s) ({@link #serializeAsField}) should be overridden
     * to change the behavior
     */
    public final Object get(Object bean) throws Exception
    {
        if (_accessorMethod != null) {
            return _accessorMethod.invoke(bean);
        }
        return _field.get(bean);
    }

    protected void _reportSelfReference(Object bean)
        throws JsonMappingException
    {
        throw new JsonMappingException("Direct self-reference leading to cycle");
    }

    @Override
    public String toString()
    {
        StringBuilder sb = new StringBuilder(40);
        sb.append("property '").append(getName()).append("' (");
        if (_accessorMethod != null) {
            sb.append("via method
").append(_accessorMethod.getDeclaringClass().getName()).append("#").append(_accessorMethod.getName());
            } else {
            sb.append("field \"").append(_field.getDeclaringClass().getName()).append("#").append(_field.getName());
```

```
        }
        sb.append(')');
        return sb.toString();
    }
}
package org.codehaus.jackson.map.ser;

import java.math.BigDecimal;
import java.math.BigInteger;
import java.net.InetAddress;
import java.util.*;

import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JsonSerialize;
import org.codehaus.jackson.map.ext.OptionalHandlerFactory;
import org.codehaus.jackson.map.introspect.Annotated;
import org.codehaus.jackson.map.introspect.AnnotatedClass;
import org.codehaus.jackson.map.introspect.AnnotatedMethod;
import org.codehaus.jackson.map.introspect.BasicBeanDescription;
import org.codehaus.jackson.map.jsontype.NamedType;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.map.ser.impl.IndexedStringListSerializer;
import org.codehaus.jackson.map.ser.impl.InetAddressSerializer;
import org.codehaus.jackson.map.ser.impl.ObjectArraySerializer;
import org.codehaus.jackson.map.ser.impl.StringCollectionSerializer;
import org.codehaus.jackson.map.ser.impl.TimeZoneSerializer;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.map.util.EnumValues;
import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.util.TokenBuffer;

/**
* Factory class that can provide serializers for standard JDK classes,
* as well as custom classes that extend standard classes or implement
* one of "well-known" interfaces (such as {@link java.util.Collection}).
*<p>
* Since all the serializers are eagerly instantiated, and there is
* no additional introspection or customizability of these types,
* this factory is essentially stateless.
*/
public abstract class BasicSerializerFactory
    extends SerializerFactory
{
    /*
    /**********************************************************
    /* Helper classes
```

```
/***************************************************************
 */

/**
 * Bogus class used to instantiate markers used as placeholders
 * for specific serializable types.
 */
private final static class SerializerMarker extends JsonSerializer<Object>
{
   @Override
   public void serialize(Object value, JsonGenerator jgen, SerializerProvider provider) { }
}

/*
/***************************************************************
/* Configuration, lookup tables/maps
/***************************************************************
 */

/**
 * Since these are all JDK classes, we shouldn't have to worry
 * about ClassLoader used to load them. Rather, we can just
 * use the class name, and keep things simple and efficient.
 */
protected final static HashMap<String, JsonSerializer<?>> _concrete =
   new HashMap<String, JsonSerializer<?>>();

/**
 * Actually it may not make much sense to eagerly instantiate all
 * kinds of serializers: so this Map actually contains class references,
 * not instances
 *
 * @since 1.6
 */
protected final static HashMap<String, Class<? extends JsonSerializer<?>>> _concreteLazy =
   new HashMap<String, Class<? extends JsonSerializer<?>>>();

// And then Java Collection classes
final static JsonSerializer<?> MARKER_INDEXED_LIST = new SerializerMarker();
final static JsonSerializer<?> MARKER_COLLECTION = new SerializerMarker();
final static JsonSerializer<?> MARKER_OBJECT_ARRAY = new SerializerMarker();
final static JsonSerializer<?> MARKER_STRING_ARRAY = new SerializerMarker();
final static JsonSerializer<?> MARKER_OBJECT_MAP = new SerializerMarker();

static {
   /* String and string-like types (note: date types explicitly
    * not included -- can use either textual or numeric serialization)
    */
```

```
_concrete.put(String.class.getName(), new StdSerializers.StringSerializer());
final ToStringSerializer sls = ToStringSerializer.instance;
_concrete.put(StringBuffer.class.getName(), sls);
_concrete.put(StringBuilder.class.getName(), sls);
_concrete.put(Character.class.getName(), sls);
_concrete.put(Character.TYPE.getName(), sls);

// Primitives/wrappers for primitives (primitives needed for Beans)
_concrete.put(Boolean.TYPE.getName(), new StdSerializers.BooleanSerializer(true));
_concrete.put(Boolean.class.getName(), new StdSerializers.BooleanSerializer(false));
final JsonSerializer<?> intS = new StdSerializers.IntegerSerializer();
_concrete.put(Integer.class.getName(), intS);
_concrete.put(Integer.TYPE.getName(), intS);
_concrete.put(Long.class.getName(), StdSerializers.LongSerializer.instance);
_concrete.put(Long.TYPE.getName(), StdSerializers.LongSerializer.instance);
_concrete.put(Byte.class.getName(), StdSerializers.IntLikeSerializer.instance);
_concrete.put(Byte.TYPE.getName(), StdSerializers.IntLikeSerializer.instance);
_concrete.put(Short.class.getName(), StdSerializers.IntLikeSerializer.instance);
_concrete.put(Short.TYPE.getName(), StdSerializers.IntLikeSerializer.instance);

// Numbers, limited length floating point
_concrete.put(Float.class.getName(), StdSerializers.FloatSerializer.instance);
_concrete.put(Float.TYPE.getName(), StdSerializers.FloatSerializer.instance);
_concrete.put(Double.class.getName(), StdSerializers.DoubleSerializer.instance);
_concrete.put(Double.TYPE.getName(), StdSerializers.DoubleSerializer.instance);

// Other numbers, more complicated
final JsonSerializer<?> ns = new StdSerializers.NumberSerializer();
_concrete.put(BigInteger.class.getName(), ns);
_concrete.put(BigDecimal.class.getName(), ns);

/* Other discrete non-container types:
 * first, Date/Time zoo:
 */
_concrete.put(Calendar.class.getName(), StdSerializers.CalendarSerializer.instance);
_concrete.put(java.util.Date.class.getName(), StdSerializers.UtilDateSerializer.instance);
_concrete.put(java.sql.Date.class.getName(), new StdSerializers.SqlDateSerializer());
_concrete.put(java.sql.Time.class.getName(), new StdSerializers.SqlTimeSerializer());
// note: timestamps are very similar to java.util.Date, thus serialized as such
_concrete.put(java.sql.Timestamp.class.getName(), StdSerializers.UtilDateSerializer.instance);

// Arrays of various types (including common object types)
_concrete.put(boolean[].class.getName(), new ArraySerializers.BooleanArraySerializer());
_concrete.put(byte[].class.getName(), new ArraySerializers.ByteArraySerializer());
_concrete.put(char[].class.getName(), new ArraySerializers.CharArraySerializer());
_concrete.put(short[].class.getName(), new ArraySerializers.ShortArraySerializer());
_concrete.put(int[].class.getName(), new ArraySerializers.IntArraySerializer());
_concrete.put(long[].class.getName(), new ArraySerializers.LongArraySerializer());
```

```
    _concrete.put(float[].class.getName(), new ArraySerializers.FloatArraySerializer());
    _concrete.put(double[].class.getName(), new ArraySerializers.DoubleArraySerializer());

    _concrete.put(Object[].class.getName(), MARKER_OBJECT_ARRAY);
    _concrete.put(String[].class.getName(), MARKER_STRING_ARRAY);

    _concrete.put(ArrayList.class.getName(), MARKER_INDEXED_LIST);
    _concrete.put(Vector.class.getName(), MARKER_INDEXED_LIST);
    _concrete.put(LinkedList.class.getName(), MARKER_COLLECTION);
    // (java.util.concurrent has others, but let's allow those to be
    // found via slower introspection; too many to enumerate here)

    _concrete.put(HashMap.class.getName(), MARKER_OBJECT_MAP);
    _concrete.put(Hashtable.class.getName(), MARKER_OBJECT_MAP);
    _concrete.put(LinkedHashMap.class.getName(), MARKER_OBJECT_MAP);
    _concrete.put(TreeMap.class.getName(), MARKER_OBJECT_MAP);
    _concrete.put(Properties.class.getName(), MARKER_OBJECT_MAP);

    _concrete.put(HashSet.class.getName(), MARKER_COLLECTION);
    _concrete.put(LinkedHashSet.class.getName(), MARKER_COLLECTION);
    _concrete.put(TreeSet.class.getName(), MARKER_COLLECTION);

    // And then other standard non-structured JDK types
    for (Map.Entry<Class<?>,Object> en : new JdkSerializers().provide()) {
        Object value = en.getValue();
        if (value instanceof JsonSerializer<?>) {
            _concrete.put(en.getKey().getName(), (JsonSerializer<?>) value);
        } else if (value instanceof Class<?>) {
            @SuppressWarnings("unchecked")
            Class<? extends JsonSerializer<?>> cls = (Class<? extends JsonSerializer<?>>) value;
            _concreteLazy.put(en.getKey().getName(), cls);
        } else { // should never happen, but:
            throw new IllegalStateException("Internal error: unrecognized value of type "+en.getClass().getName());
        }
    }

    // Jackson-specific type(s)
    // (Q: can this ever be sub-classed?)
    _concreteLazy.put(TokenBuffer.class.getName(), StdSerializers.TokenBufferSerializer.class);
}

/**
 * Helper object used to deal with serializers for optional JDK types (like ones
 * omitted from GAE, Android)
 */
protected OptionalHandlerFactory optionalHandlers = OptionalHandlerFactory.instance;

/*
```

```
/**********************************************************
/* Life cycle
/**********************************************************
 */

/**
 * We will provide default constructor to allow sub-classing,
 * but make it protected so that no non-singleton instances of
 * the class will be instantiated.
 */
protected BasicSerializerFactory() { }

/*
/**********************************************************
/* SerializerFactory impl
/**********************************************************
 */

/**
 * Main serializer constructor method. The base implementation within
 * this class first calls a fast lookup method that can find serializers
 * for well-known JDK classes; and if that fails, a slower one that
 * tries to check out which interfaces given Class implements.
 * Sub-classes can (and do) change this behavior to alter behavior.
 */
@Override
public JsonSerializer<Object> createSerializer(SerializationConfig config, JavaType type,
        BeanProperty property)
{
    /* [JACKSON-220]: Very first thing, let's check annotations to
     * see if we have explicit definition
     */
    BasicBeanDescription beanDesc = config.introspect(type);
    JsonSerializer<?> ser = findSerializerFromAnnotation(config, beanDesc.getClassInfo(), property);
    if (ser == null) {
        // First, fast lookup for exact type:
        ser = findSerializerByLookup(type, config, beanDesc, property);
        if (ser == null) {
            /* and should that fail, slower introspection methods; first
             * one that deals with "primary" types
             */
            ser = findSerializerByPrimaryType(type, config, beanDesc, property);
            if (ser == null) {
                // And if that fails, one with "secondary" traits:
                ser = findSerializerByAddonType(config, type, beanDesc, property);
            }
        }
    }
```

```
    @SuppressWarnings("unchecked")
    JsonSerializer<Object> s2 = (JsonSerializer<Object>) ser;
    return s2;
}

/**
 * Method called to construct a type serializer for values with given declared
 * base type. This is called for values other than those of bean property
 * types.
 */
@Override
public TypeSerializer createTypeSerializer(SerializationConfig config, JavaType baseType,
        BeanProperty property)
{
    BasicBeanDescription bean = config.introspectClassAnnotations(baseType.getRawClass());
    AnnotatedClass ac = bean.getClassInfo();
    AnnotationIntrospector ai = config.getAnnotationIntrospector();
    TypeResolverBuilder<?> b = ai.findTypeResolver(ac, baseType);
    /* Ok: if there is no explicit type info handler, we may want to
     * use a default. If so, config object knows what to use.
     */
    Collection<NamedType> subtypes = null;
    if (b == null) {
        b = config.getDefaultTyper(baseType);
    } else {
        subtypes = config.getSubtypeResolver().collectAndResolveSubtypes(ac, config, ai);
    }
    return (b == null) ? null : b.buildTypeSerializer(baseType, subtypes, property);
}


/*
/**********************************************************
/* Additional API for other core classes
/**********************************************************
 */

public final JsonSerializer<?> getNullSerializer() {
    return NullSerializer.instance;
}

/*
/**********************************************************
/* Overridable secondary serializer accessor methods
/**********************************************************
 */

/**
```

```
     * Fast lookup-based accessor method, which will only check for
     * type itself, but not consider super-classes or implemented
     * interfaces.
     */
    public final JsonSerializer<?> findSerializerByLookup(JavaType type, SerializationConfig config,
            BasicBeanDescription beanDesc, BeanProperty property)
    {
        String clsName = type.getRawClass().getName();
        JsonSerializer<?> ser = _concrete.get(clsName);
        if (ser == null) {
            Class<? extends JsonSerializer<?>> serClass = _concreteLazy.get(clsName);
            if (serClass != null) {
                try {
                    ser = serClass.newInstance();
                } catch (Exception e) {
                    throw new IllegalStateException("Failed to instantiate standard serializer (of type
"+serClass.getName()+"): "
                            +e.getMessage(), e);
                }
            }
        }

        /* 08-Nov-2009, tatus: Some standard types may need customization;
         *    for now that just means Maps, but in future probably other
         *    collections as well. For strictly standard types this is
         *    currently only needed due to mix-in annotations.
         */
        if (ser != null ) {
            if (ser == MARKER_OBJECT_MAP) {
                return buildMapSerializer(config, type, beanDesc, property);
            }
            if (ser == MARKER_OBJECT_ARRAY) {
                return buildObjectArraySerializer(config, type, beanDesc, property);
            }
            if (ser == MARKER_STRING_ARRAY) {
                return new ArraySerializers.StringArraySerializer(property);
            }
            if (ser == MARKER_INDEXED_LIST) {
                // Ok: for some types we have specialized handlers
                JavaType elemType = type.getContentType();
                if (elemType.getRawClass() == String.class) {
                    return new IndexedStringListSerializer(property);
                }
                return buildIndexedListSerializer(config, type, beanDesc, property);
            }
            if (ser == MARKER_COLLECTION) {
                // Ok: for some types we have specialized handlers
                JavaType elemType = type.getContentType();
```

```
            if (elemType.getRawClass() == String.class) {
                return new StringCollectionSerializer(property);
            }
            return buildCollectionSerializer(config, type, beanDesc, property);
        }
    } else {
        Class<?> raw = type.getRawClass();
        // One unfortunate special case, as per [JACKSON-484]
        if (InetAddress.class.isAssignableFrom(raw)) {
            return InetAddressSerializer.instance;
        }
        // ... and another one, [JACKSON-522], for TimeZone
        if (TimeZone.class.isAssignableFrom(raw)) {
            return TimeZoneSerializer.instance;
        }

        // Then check for optional/external serializers [JACKSON-386]
        ser = optionalHandlers.findSerializer(config, type, beanDesc, property);
    }
    return ser;
}

/**
 * Reflection-based serialized find method, which checks if
 * given class is a sub-type of one of well-known classes, or implements
 * a "primary" interface. Primary here is defined as the main function
 * of the Object; as opposed to "add-on" functionality.
 */
@SuppressWarnings("deprecation")
public final JsonSerializer<?> findSerializerByPrimaryType(JavaType type, SerializationConfig config,
        BasicBeanDescription beanDesc, BeanProperty property)
{
    Class<?> cls = type.getRawClass();

    /* Some types are final, and hence not checked here (will
     * have been handled by fast method above):
     *
     * - Boolean
     * - String (StringBuffer, StringBuilder)
     * - Arrays for primitive types
     *
     * But we do need to check for
     *
     * - "primary" interfaces: Enum, Number, JsonSerializable
     * - Most collection types
     * - java.lang.Number (but is that integral or not?)
     */
    if (JsonSerializable.class.isAssignableFrom(cls)) {
```

```
    if (JsonSerializableWithType.class.isAssignableFrom(cls)) {
        return StdSerializers.SerializableWithTypeSerializer.instance;
    }
    return StdSerializers.SerializableSerializer.instance;
}
if (Map.class.isAssignableFrom(cls)) {
    if (EnumMap.class.isAssignableFrom(cls)) {
        return buildEnumMapSerializer(config, type, beanDesc, property);
    }
    return buildMapSerializer(config, type, beanDesc, property);
}
if (Object[].class.isAssignableFrom(cls)) {
    return buildObjectArraySerializer(config, type, beanDesc, property);
}
if (List.class.isAssignableFrom(cls)) {
    if (cls == List.class || cls == AbstractList.class || RandomAccess.class.isAssignableFrom(cls)) {
        return buildIndexedListSerializer(config, type, beanDesc, property);
    }
    return buildCollectionSerializer(config, type, beanDesc, property);
}
// [JACKSON-193]: consider @JsonValue for enum types (and basically any type), so:
AnnotatedMethod valueMethod = beanDesc.findJsonValueMethod();
if (valueMethod != null) {
    JsonSerializer<Object> ser = findSerializerFromAnnotation(config, valueMethod, property);
    return new JsonValueSerializer(valueMethod.getAnnotated(), ser, property);
}

if (Number.class.isAssignableFrom(cls)) {
    return StdSerializers.NumberSerializer.instance;
}
if (Enum.class.isAssignableFrom(cls)) {
    @SuppressWarnings("unchecked")
    Class<Enum<?>> enumClass = (Class<Enum<?>>) cls;
    return EnumSerializer.construct(enumClass, config, beanDesc);
}
if (Calendar.class.isAssignableFrom(cls)) {
    return StdSerializers.CalendarSerializer.instance;
}
if (java.util.Date.class.isAssignableFrom(cls)) {
    return StdSerializers.UtilDateSerializer.instance;
}
if (Collection.class.isAssignableFrom(cls)) {
    if (EnumSet.class.isAssignableFrom(cls)) {
        return buildEnumSetSerializer(config, type, beanDesc, property);
    }
    return buildCollectionSerializer(config, type, beanDesc, property);
}
return null;
```

```
    }

    /**
     * Reflection-based serialized find method, which checks if
     * given class implements one of recognized "add-on" interfaces.
     * Add-on here means a role that is usually or can be a secondary
     * trait: for example,
     * bean classes may implement {@link Iterable}, but their main
     * function is usually something else. The reason for
     */
    public final JsonSerializer<?> findSerializerByAddonType(SerializationConfig config, JavaType javaType,
            BasicBeanDescription beanDesc, BeanProperty property)
    {
        Class<?> type = javaType.getRawClass();

        // These need to be in decreasing order of specificity...
        if (Iterator.class.isAssignableFrom(type)) {
            return buildIteratorSerializer(config, javaType, beanDesc, property);
        }
        if (Iterable.class.isAssignableFrom(type)) {
            return buildIterableSerializer(config, javaType, beanDesc, property);
        }
        if (CharSequence.class.isAssignableFrom(type)) {
            return ToStringSerializer.instance;
        }
        return null;
    }

    /**
     * Helper method called to check if a class or method
     * has an annotation
     * (@link org.codehaus.jackson.map.ser.JsonSerialize#using)
     * that tells the class to use for serialization.
     * Returns null if no such annotation found.
     */
    @SuppressWarnings("unchecked")
    protected JsonSerializer<Object> findSerializerFromAnnotation(SerializationConfig config, Annotated a,
BeanProperty property)
    {
        Object serDef = config.getAnnotationIntrospector().findSerializer(a, property);
        if (serDef != null) {
            if (serDef instanceof JsonSerializer) {
                return (JsonSerializer<Object>) serDef;
            }
            /* Alas, there's no way to force return type of "either class
             * X or Y" -- need to throw an exception after the fact
             */
            if (!(serDef instanceof Class)) {
```

```
            throw new IllegalStateException("AnnotationIntrospector returned value of type
"+serDef.getClass().getName()+"; expected type JsonSerializer or Class<JsonSerializer> instead");
        }
        Class<?> cls = (Class<?>) serDef;
        if (!JsonSerializer.class.isAssignableFrom(cls)) {
            throw new IllegalStateException("AnnotationIntrospector returned Class "+cls.getName()+"; expected
Class<JsonSerializer>");
        }
        return (JsonSerializer<Object>) ClassUtil.createInstance(cls,
config.isEnabled(SerializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS));
    }
    return null;
  }

  /**
   * Helper method that handles configuration details when constructing serializers for
   * {@link java.util.Map} types.
   */
  protected JsonSerializer<?> buildMapSerializer(SerializationConfig config, JavaType type,
        BasicBeanDescription beanDesc, BeanProperty property)
  {
    AnnotationIntrospector intr = config.getAnnotationIntrospector();
    JavaType valueType = type.getContentType();
    TypeSerializer vts = createTypeSerializer(config, valueType, property);
    boolean staticTyping = usesStaticTyping(config, beanDesc, vts);
    return MapSerializer.construct(intr.findPropertiesToIgnore(beanDesc.getClassInfo()),
            type, staticTyping, vts, property);
  }

  protected JsonSerializer<?> buildEnumMapSerializer(SerializationConfig config, JavaType type,
        BasicBeanDescription beanDesc, BeanProperty property)
  {
    JavaType keyType = type.getKeyType();
    JavaType valueType = type.getContentType();
    // Need to find key enum values...
    EnumValues enums = null;
    if (keyType.isEnumType()) { // non-enum if we got it as type erased class (from instance)
      @SuppressWarnings("unchecked")
      Class<Enum<?>> enumClass = (Class<Enum<?>>) keyType.getRawClass();
      enums = EnumValues.construct(enumClass, config.getAnnotationIntrospector());
    }
    TypeSerializer vts = createTypeSerializer(config, valueType, property);
    return new EnumMapSerializer(valueType, usesStaticTyping(config, beanDesc, vts),
            enums, vts, property);
  }

  /**
   * Helper method that handles configuration details when constructing serializers for
```

```
     * <code>Object[]</code> (and subtypes, except for String).
     */
    protected JsonSerializer<?> buildObjectArraySerializer(SerializationConfig config, JavaType type,
            BasicBeanDescription beanDesc, BeanProperty property)
    {
        JavaType valueType = type.getContentType();
        TypeSerializer vts = createTypeSerializer(config, valueType, property);
        return new ObjectArraySerializer(valueType,
                usesStaticTyping(config, beanDesc, vts), vts, property);
    }

    protected JsonSerializer<?> buildIndexedListSerializer(SerializationConfig config, JavaType type,
            BasicBeanDescription beanDesc, BeanProperty property)
    {
        JavaType valueType = type.getContentType();
        TypeSerializer vts = createTypeSerializer(config, valueType, property);
        return ContainerSerializers.indexedListSerializer(valueType,
                usesStaticTyping(config, beanDesc, vts), vts, property);
    }

    protected JsonSerializer<?> buildCollectionSerializer(SerializationConfig config, JavaType type,
            BasicBeanDescription beanDesc, BeanProperty property)
    {
        JavaType valueType = type.getContentType();
        TypeSerializer vts = createTypeSerializer(config, valueType, property);
        return ContainerSerializers.collectionSerializer(valueType,
                usesStaticTyping(config, beanDesc, vts), vts, property);
    }

    protected JsonSerializer<?> buildIteratorSerializer(SerializationConfig config, JavaType type,
            BasicBeanDescription beanDesc, BeanProperty property)
    {
        // if there's generic type, it'll be the first contained type
        JavaType valueType = type.containedType(0);
        if (valueType == null) {
            valueType = TypeFactory.type(Object.class);
        }
        TypeSerializer vts = createTypeSerializer(config, valueType, property);
        return ContainerSerializers.iteratorSerializer(valueType,
                usesStaticTyping(config, beanDesc, vts), vts, property);
    }

    protected JsonSerializer<?> buildIterableSerializer(SerializationConfig config, JavaType type,
            BasicBeanDescription beanDesc, BeanProperty property)
    {
        // if there's generic type, it'll be the first contained type
        JavaType valueType = type.containedType(0);
        if (valueType == null) {
```

```
            valueType = TypeFactory.type(Object.class);
        }
        TypeSerializer vts = createTypeSerializer(config, valueType, property);
        return ContainerSerializers.iterableSerializer(valueType,
                usesStaticTyping(config, beanDesc, vts), vts, property);
    }

    protected JsonSerializer<?> buildEnumSetSerializer(SerializationConfig config, JavaType type,
            BasicBeanDescription beanDesc, BeanProperty property)
    {
        // this may or may not be available (Class doesn't; type of field/method does)
        JavaType enumType = type.getContentType();
        // and even if nominally there is something, only use if it really is enum
        if (!enumType.isEnumType()) {
            enumType = null;
        }
        return ContainerSerializers.enumSetSerializer(enumType, property);
    }

    /**
     * Helper method to check whether global settings and/or class
     * annotations for the bean class indicate that static typing
     * (declared types)  should be used for properties.
     * (instead of dynamic runtime types).
     */
    protected boolean usesStaticTyping(SerializationConfig config, BasicBeanDescription beanDesc,
            TypeSerializer typeSer)
    {
        /* 16-Aug-2010, tatu: If there is a (value) type serializer, we can not force
         *    static typing; that would make it impossible to handle expected subtypes
         *
         */
        if (typeSer != null) {
            return false;
        }
        JsonSerialize.Typing t = config.getAnnotationIntrospector().findSerializationTyping(beanDesc.getClassInfo());
        if (t != null) {
            return (t == JsonSerialize.Typing.STATIC);
        }
        return config.isEnabled(SerializationConfig.Feature.USE_STATIC_TYPING);
    }
}
package org.codehaus.jackson.map.ser;

import java.util.*;

import org.codehaus.jackson.annotate.JsonAutoDetect.Visibility;
import org.codehaus.jackson.map.*;
```

```
import org.codehaus.jackson.map.introspect.*;
import org.codehaus.jackson.map.jsontype.NamedType;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.map.type.TypeBindings;
import org.codehaus.jackson.map.util.ArrayBuilders;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.type.JavaType;

/**
* Factory class that can provide serializers for any regular Java beans
* (as defined by "having at least one get method recognizable as bean
* accessor" -- where {@link Object#getClass} does not count);
* as well as for "standard" JDK types. Latter is achieved
* by delegating calls to {@link BasicSerializerFactory}
* to find serializers both for "standard" JDK types (and in some cases,
* sub-classes as is the case for collection classes like
* {@link java.util.List}s and {@link java.util.Map}s) and bean (value)
* classes.
*<p>
* Note about delegating calls to {@link BasicSerializerFactory}:
* although it would be nicer to use linear delegation
* for construction (to essentially dispatch all calls first to the
* underlying {@link BasicSerializerFactory}; or alternatively after
* failing to provide bean-based serializer}, there is a problem:
* priority levels for detecting standard types are mixed. That is,
* we want to check if a type is a bean after some of "standard" JDK
* types, but before the rest.
* As a result, "mixed" delegation used, and calls are NOT done using
* regular {@link SerializerFactory} interface but rather via
* direct calls to {@link BasicSerializerFactory}.
*<p>
* Finally, since all caching is handled by the serializer provider
* (not factory) and there is no configurability, this
* factory is stateless.
* This means that a global singleton instance can be used.
*<p>
* Notes for version 1.7 (and above): the new module registration system
* required addition of {@link #withConfig}, which has to
* be redefined by sub-classes so that they can work properly with
* pluggable additional serializer providing components.
*/
public class BeanSerializerFactory
  extends BasicSerializerFactory
{
  /**
   * Like {@link BasicSerializerFactory}, this factory is stateless, and
   * thus a single shared global (== singleton) instance can be used
   * without thread-safety issues.
```

```
 */
public final static BeanSerializerFactory instance = new BeanSerializerFactory(null);

/**
 * Configuration settings for this factory; immutable instance (just like this
 * factory), new version created via copy-constructor (fluent-style)
 *
 * @since 1.7
 */
protected final Config _factoryConfig;

/*
/**********************************************************
/* Config class implementation
/**********************************************************
 */

/**
 * Configuration settings container class for bean serializer factory
 *
 * @since 1.7
 */
public static class ConfigImpl extends Config
{
    /**
     * Constant for empty <code>Serializers</code> array (which by definition
     * is stateless and reusable)
     */
    protected final static Serializers[] NO_SERIALIZERS = new Serializers[0];

    protected final static BeanSerializerModifier[] NO_MODIFIERS = new BeanSerializerModifier[0];

    /**
     * List of providers for additional serializers, checked before considering default
     * basic or bean serialializers.
     *
     * @since 1.7
     */
    protected final Serializers[] _additionalSerializers;

    /**
     * List of modifiers that can change the way {@link BeanSerializer} instances
     * are configured and constructed.
     */
    protected final BeanSerializerModifier[] _modifiers;

    public ConfigImpl() {
        this(null, null);
```

```
    }

    protected ConfigImpl(Serializers[] allAdditionalSerializers,
            BeanSerializerModifier[] modifiers)
    {
        _additionalSerializers = (allAdditionalSerializers == null) ?
                NO_SERIALIZERS : allAdditionalSerializers;
        _modifiers = (modifiers == null) ? NO_MODIFIERS : modifiers;
    }

    @Override
    public Config withAdditionalSerializers(Serializers additional)
    {
        if (additional == null) {
            throw new IllegalArgumentException("Can not pass null Serializers");
        }
        Serializers[] all = ArrayBuilders.insertInList(_additionalSerializers, additional);
        return new ConfigImpl(all, _modifiers);
    }

    @Override
    public Config withSerializerModifier(BeanSerializerModifier modifier)
    {
        if (modifier == null) {
            throw new IllegalArgumentException("Can not pass null modifier");
        }
        BeanSerializerModifier[] modifiers = ArrayBuilders.insertInList(_modifiers, modifier);
        return new ConfigImpl(_additionalSerializers, modifiers);
    }

    @Override
    public boolean hasSerializers() { return _additionalSerializers.length > 0; }

    @Override
    public boolean hasSerializerModifiers() { return _modifiers.length > 0; }

    @Override
    public Iterable<Serializers> serializers() {
        return ArrayBuilders.arrayAsIterable(_additionalSerializers);
    }

    @Override
    public Iterable<BeanSerializerModifier> serializerModifiers() {
        return ArrayBuilders.arrayAsIterable(_modifiers);
    }
}

/*
```

```
/**********************************************************
/* Life-cycle: creation, configuration
/**********************************************************
 */

@Deprecated
protected BeanSerializerFactory() { this(null); }

/**
 * Constructor for creating instances with specified configuration.
 */
protected BeanSerializerFactory(Config config)
{
   if (config == null) {
      config = new ConfigImpl();
   }
   _factoryConfig = config;
}

@Override public Config getConfig() { return _factoryConfig; }

/**
 * Method used by module registration functionality, to attach additional
 * serializer providers into this serializer factory. This is typically
 * handled by constructing a new instance with additional serializers,
 * to ensure thread-safe access.
 *
 * @since 1.7
 */
@Override
public SerializerFactory withConfig(Config config)
{
   if (_factoryConfig == config) {
      return this;
   }
   /* 22-Nov-2010, tatu: Handling of subtypes is tricky if we do immutable-with-copy-ctor;
    *    and we pretty much have to here either choose between losing subtype instance
    *    when registering additional serializers, or losing serializers.
    *    Instead, let's actually just throw an error if this method is called when subtype
    *    has not properly overridden this method; this to indicate problem as soon as possible.
    */
   if (getClass() != BeanSerializerFactory.class) {
      throw new IllegalStateException("Subtype of BeanSerializerFactory ("+getClass().getName()
            +") has not properly overridden method 'withAdditionalSerializers': can not instantiate subtype with "
            +"additional serializer definitions");
   }
   return new BeanSerializerFactory(config);
}
```

```
/*
/**********************************************************
/* JsonSerializerFactory impl
/**********************************************************
 */

/**
 * Main serializer constructor method. We will have to be careful
 * with respect to ordering of various method calls: essentially
 * we want to reliably figure out which classes are standard types,
 * and which are beans. The problem is that some bean Classes may
 * implement standard interfaces (say, {@link java.lang.Iterable}.
 *<p>
 * Note: sub-classes may choose to complete replace implementation,
 * if they want to alter priority of serializer lookups.
 */
@Override
@SuppressWarnings("unchecked")
public JsonSerializer<Object> createSerializer(SerializationConfig config, JavaType type,
      BeanProperty property)
{
   /* [JACKSON-220]: Very first thing, let's check annotations to
    * see if we have explicit definition
    */
   BasicBeanDescription beanDesc = config.introspect(type);
   JsonSerializer<?> ser = findSerializerFromAnnotation(config, beanDesc.getClassInfo(), property);
   if (ser == null) {
      // 22-Nov-2010, tatu: Ok: additional module-provided serializers to consider?
      ser = _findFirstSerializer(_factoryConfig.serializers(), config, type, beanDesc, property);
      if (ser == null) {
         // First, fast lookup for exact type:
         ser = super.findSerializerByLookup(type, config, beanDesc, property);
         if (ser == null) {
            // and then introspect for some safe (?) JDK types
            ser = super.findSerializerByPrimaryType(type, config, beanDesc, property);
            if (ser == null) {
               /* And this is where this class comes in: if type is
                * not a known "primary JDK type", perhaps it's a bean?
                * We can still get a null, if we can't find a single
                * suitable bean property.
                */
               ser = this.findBeanSerializer(config, type, beanDesc, property);
               /* Finally: maybe we can still deal with it as an
                * implementation of some basic JDK interface?
                */
               if (ser == null) {
                  ser = super.findSerializerByAddonType(config, type, beanDesc, property);
```

```
                }
            }
        }
    }
    return (JsonSerializer<Object>) ser;
}

/**
 * Helper method used to try to find serializer from set of registered
 * {@link Serializers} instances (provided by registered Modules),
 * and return first one found, if any.
 */
private static JsonSerializer<?> _findFirstSerializer(Iterable<Serializers> sers, SerializationConfig config,
        JavaType type, BeanDescription beanDesc, BeanProperty property)
{
    for (Serializers ser : sers) {
        JsonSerializer<?> js = ser.findSerializer(config, type, beanDesc, property);
        if (js != null) {
            return js;
        }
    }
    return null;
}

/*
/**********************************************************
/* Other public methods that are not part of
/* JsonSerializerFactory API
/**********************************************************
 */

/**
 * Method that will try to construct a {@link BeanSerializer} for
 * given class. Returns null if no properties are found.
 */
@SuppressWarnings("unchecked")
public JsonSerializer<Object> findBeanSerializer(SerializationConfig config, JavaType type,
        BasicBeanDescription beanDesc, BeanProperty property)
{
    // First things first: we know some types are not beans...
    if (!isPotentialBeanType(type.getRawClass())) {
        return null;
    }
    JsonSerializer<Object> serializer = constructBeanSerializer(config, beanDesc, property);
    // [JACKSON-440] Need to allow overriding actual serializer, as well...
    if (_factoryConfig.hasSerializerModifiers()) {
        for (BeanSerializerModifier mod : _factoryConfig.serializerModifiers()) {
```

```
                serializer = (JsonSerializer<Object>)mod.modifySerializer(config, beanDesc, serializer);
            }
        }
        return serializer;
    }

    /**
     * Method called to create a type information serializer for values of given
     * non-container property
     * if one is needed. If not needed (no polymorphic handling configured), should
     * return null.
     *
     * @param baseType Declared type to use as the base type for type information serializer
     *
     * @return Type serializer to use for property values, if one is needed; null if not.
     *
     * @since 1.5
     */
    public TypeSerializer findPropertyTypeSerializer(JavaType baseType, SerializationConfig config,
            AnnotatedMember accessor, BeanProperty property)
    {
        AnnotationIntrospector ai = config.getAnnotationIntrospector();
        TypeResolverBuilder<?> b = ai.findPropertyTypeResolver(accessor, baseType);
        // Defaulting: if no annotations on member, check value class
        if (b == null) {
            return createTypeSerializer(config, baseType, property);
        }
        Collection<NamedType> subtypes = config.getSubtypeResolver().collectAndResolveSubtypes(accessor,
config, ai);
        return b.buildTypeSerializer(baseType, subtypes, property);
    }

    /**
     * Method called to create a type information serializer for values of given
     * container property
     * if one is needed. If not needed (no polymorphic handling configured), should
     * return null.
     *
     * @param containerType Declared type of the container to use as the base type for type information serializer
     *
     * @return Type serializer to use for property value contents, if one is needed; null if not.
     *
     * @since 1.5
     */
    public TypeSerializer findPropertyContentTypeSerializer(JavaType containerType, SerializationConfig config,
            AnnotatedMember accessor, BeanProperty property)
    {
        JavaType contentType = containerType.getContentType();
```

```
        AnnotationIntrospector ai = config.getAnnotationIntrospector();
        TypeResolverBuilder<?> b = ai.findPropertyContentTypeResolver(accessor, containerType);
        // Defaulting: if no annotations on member, check value class
        if (b == null) {
            return createTypeSerializer(config, contentType, property);
        }
        Collection<NamedType> subtypes = config.getSubtypeResolver().collectAndResolveSubtypes(accessor,
config, ai);
        return b.buildTypeSerializer(contentType, subtypes, property);
    }

    /*
    /**********************************************************
    /* Overridable non-public factory methods
    /**********************************************************
     */

    /**
     * Method called to construct serializer for serializing specified bean type.
     *
     * @since 1.6
     */
    @SuppressWarnings("unchecked")
    protected JsonSerializer<Object> constructBeanSerializer(SerializationConfig config,
            BasicBeanDescription beanDesc, BeanProperty property)
    {
        // 13-Oct-2010, tatu: quick sanity check: never try to create bean serializer for plain Object
        if (beanDesc.getBeanClass() == Object.class) {
            throw new IllegalArgumentException("Can not create bean serializer for Object.class");
        }

        BeanSerializerBuilder builder = constructBeanSerializerBuilder(beanDesc);

        // First: any detectable (auto-detect, annotations) properties to serialize?
        List<BeanPropertyWriter> props = findBeanProperties(config, beanDesc);
        AnnotatedMethod anyGetter = beanDesc.findAnyGetter();

        // [JACKSON-440] Need to allow modification bean properties to serialize:
        if (_factoryConfig.hasSerializerModifiers()) {
            if (props == null) {
                props = new ArrayList<BeanPropertyWriter>();
            }
            for (BeanSerializerModifier mod : _factoryConfig.serializerModifiers()) {
                props = mod.changeProperties(config, beanDesc, props);
            }
        }

        // No properties, no serializer
```

```
// 16-Oct-2010, tatu: Except that @JsonAnyGetter needs to count as getter
if (props == null || props.size() == 0) {
    if (anyGetter == null) {
        /* 27-Nov-2009, tatu: Except that as per [JACKSON-201], we are
         *   ok with that as long as it has a recognized class annotation
         *  (which may come from a mix-in too)
         */
        if (beanDesc.hasKnownClassAnnotations()) {
            return builder.createDummy();
        }
        return null;
    }
    props = Collections.emptyList();
} else {
    // Any properties to suppress?
    props = filterBeanProperties(config, beanDesc, props);
    // Do they need to be sorted in some special way?
    props = sortBeanProperties(config, beanDesc, props);
}
// [JACKSON-440] Need to allow reordering of properties to serialize
if (_factoryConfig.hasSerializerModifiers()) {
    for (BeanSerializerModifier mod : _factoryConfig.serializerModifiers()) {
        props = mod.orderProperties(config, beanDesc, props);
    }
}
builder.setProperties(props);
builder.setFilterId(findFilterId(config, beanDesc));

if (anyGetter != null) { // since 1.6
    JavaType type = anyGetter.getType(beanDesc.bindingsForBeanType());
    // copied from BasicSerializerFactory.buildMapSerializer():
    boolean staticTyping = config.isEnabled(SerializationConfig.Feature.USE_STATIC_TYPING);
    JavaType valueType = type.getContentType();
    TypeSerializer typeSer = createTypeSerializer(config, valueType, property);
    MapSerializer mapSer = MapSerializer.construct(/* ignored props*/ null, type, staticTyping,
            typeSer, property);
    builder.setAnyGetter(new AnyGetterWriter(anyGetter, mapSer));
}
// One more thing: need to gather view information, if any:
processViews(config, builder);
// And maybe let interested parties mess with the result bit more...
if (_factoryConfig.hasSerializerModifiers()) {
    for (BeanSerializerModifier mod : _factoryConfig.serializerModifiers()) {
        builder = mod.updateBuilder(config, beanDesc, builder);
    }
}
// And finally construct serializer
return (JsonSerializer<Object>) builder.build();
```

```
}

/**
 * Method called to construct a filtered writer, for given view
 * definitions. Default implementation constructs filter that checks
 * active view type to views property is to be included in.
 */
protected BeanPropertyWriter constructFilteredBeanWriter(BeanPropertyWriter writer, Class<?>[] inViews)
{
   return FilteredBeanPropertyWriter.constructViewBased(writer, inViews);
}

protected PropertyBuilder constructPropertyBuilder(SerializationConfig config,
                                   BasicBeanDescription beanDesc)
{
   return new PropertyBuilder(config, beanDesc);
}

protected BeanSerializerBuilder constructBeanSerializerBuilder(BasicBeanDescription beanDesc) {
   return new BeanSerializerBuilder(beanDesc);
}

/**
 * Method called to find filter that is configured to be used with bean
 * serializer being built, if any.
 *
 * @since 1.7
 */
protected Object findFilterId(SerializationConfig config, BasicBeanDescription beanDesc)
{
   return config.getAnnotationIntrospector().findFilterId(beanDesc.getClassInfo());
}

/*
/**********************************************************
/* Overridable non-public introspection methods
/**********************************************************
 */

/**
 * Helper method used to skip processing for types that we know
 * can not be (i.e. are never consider to be) beans:
 * things like primitives, Arrays, Enums, and proxy types.
 *<p>
 * Note that usually we shouldn't really be getting these sort of
 * types anyway; but better safe than sorry.
 */
protected boolean isPotentialBeanType(Class<?> type)
```

```
    {
        return (ClassUtil.canBeABeanType(type) == null) && !ClassUtil.isProxyType(type);
    }

    /**
     * Method used to collect all actual serializable properties.
     * Can be overridden to implement custom detection schemes.
     */
    protected List<BeanPropertyWriter> findBeanProperties(SerializationConfig config, BasicBeanDescription
beanDesc)
    {
        // Ok: let's aggregate visibility settings: first, baseline:
        VisibilityChecker<?> vchecker = config.getDefaultVisibilityChecker();
        if (!config.isEnabled(SerializationConfig.Feature.AUTO_DETECT_GETTERS)) {
            vchecker = vchecker.withGetterVisibility(Visibility.NONE);
        }
        // then global overrides (disabling)
        if (!config.isEnabled(SerializationConfig.Feature.AUTO_DETECT_IS_GETTERS)) {
            vchecker = vchecker.withIsGetterVisibility(Visibility.NONE);
        }
        if (!config.isEnabled(SerializationConfig.Feature.AUTO_DETECT_FIELDS)) {
            vchecker = vchecker.withFieldVisibility(Visibility.NONE);
        }
        // and finally per-class overrides:
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
        vchecker = intr.findAutoDetectVisibility(beanDesc.getClassInfo(), vchecker);

        LinkedHashMap<String,AnnotatedMethod> methodsByProp = beanDesc.findGetters(vchecker, null);
        LinkedHashMap<String,AnnotatedField> fieldsByProp = beanDesc.findSerializableFields(vchecker,
methodsByProp.keySet());

        // [JACKSON-429]: ignore specified types
        removeIgnorableTypes(config, beanDesc, methodsByProp);
        removeIgnorableTypes(config, beanDesc, fieldsByProp);

        // nothing? can't proceed (caller may or may not throw an exception)
        if (methodsByProp.isEmpty() && fieldsByProp.isEmpty()) {
            return null;
        }

        // null is for value type serializer, which we don't have access to from here
        boolean staticTyping = usesStaticTyping(config, beanDesc, null);
        PropertyBuilder pb = constructPropertyBuilder(config, beanDesc);

        ArrayList<BeanPropertyWriter> props = new ArrayList<BeanPropertyWriter>(methodsByProp.size());
        TypeBindings typeBind = beanDesc.bindingsForBeanType();
        // [JACKSON-98]: start with field properties, if any
        for (Map.Entry<String,AnnotatedField> en : fieldsByProp.entrySet()) {
```

```
            // [JACKSON-235]: suppress writing of back references
            AnnotationIntrospector.ReferenceProperty prop = intr.findReferenceType(en.getValue());
            if (prop != null && prop.isBackReference()) {
                continue;
            }
            props.add(_constructWriter(config, typeBind, pb, staticTyping, en.getKey(), en.getValue()));
        }
        // and then add member properties
        for (Map.Entry<String,AnnotatedMethod> en : methodsByProp.entrySet()) {
            // [JACKSON-235]: suppress writing of back references
            AnnotationIntrospector.ReferenceProperty prop = intr.findReferenceType(en.getValue());
            if (prop != null && prop.isBackReference()) {
                continue;
            }
            props.add(_constructWriter(config, typeBind, pb, staticTyping, en.getKey(), en.getValue()));
        }
        return props;
    }

    /*
    /**********************************************************
    /* Overridable non-public methods for manipulating bean properties
    /**********************************************************
     */

    /**
     * Overridable method that can filter out properties. Default implementation
     * checks annotations class may have.
     */
    protected List<BeanPropertyWriter> filterBeanProperties(SerializationConfig config,
            BasicBeanDescription beanDesc, List<BeanPropertyWriter> props)
    {
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
        AnnotatedClass ac = beanDesc.getClassInfo();
        String[] ignored = intr.findPropertiesToIgnore(ac);
        if (ignored != null && ignored.length > 0) {
            HashSet<String> ignoredSet = ArrayBuilders.arrayToSet(ignored);
            Iterator<BeanPropertyWriter> it = props.iterator();
            while (it.hasNext()) {
                if (ignoredSet.contains(it.next().getName())) {
                    it.remove();
                }
            }
        }
        return props;
    }

    /**
```

```
     * Overridable method that will impose given partial ordering on
     * list of discovered propertied. Method can be overridden to
     * provide custom ordering of properties, beyond configurability
     * offered by annotations (whic allow alphabetic ordering, as
     * well as explicit ordering by providing array of property names).
     *<p>
     * By default Creator properties will be ordered before other
     * properties. Explicit custom ordering will override this implicit
     * default ordering.
     */
    protected List<BeanPropertyWriter> sortBeanProperties(SerializationConfig config,
            BasicBeanDescription beanDesc, List<BeanPropertyWriter> props)
    {
        // Ok: so far so good. But do we need to (re)order these somehow?
        /* Yes; first, for [JACKSON-90] (explicit ordering and/or alphabetic)
         * and then for [JACKSON-170] (implicitly order creator properties before others)
         */
        List<String> creatorProps = beanDesc.findCreatorPropertyNames();
        // Then how about explicit ordering?
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
        AnnotatedClass ac = beanDesc.getClassInfo();
        String[] propOrder = intr.findSerializationPropertyOrder(ac);
        Boolean alpha = intr.findSerializationSortAlphabetically(ac);
        boolean sort = (alpha != null) && alpha.booleanValue();
        if (sort || !creatorProps.isEmpty() || propOrder != null) {
            props = _sortBeanProperties(props, creatorProps, propOrder, sort);
        }
        return props;
    }

    /**
     * Method called to handle view information for constructed serializer,
     * based on bean property writers.
     *<p>
     * Note that this method is designed to be overridden by sub-classes
     * if they want to provide custom view handling. As such it is not
     * considered an internal implementation detail, and will be supported
     * as part of API going forward.
     *<p>
     * NOTE: signature of this method changed in 1.7, due to other significant
     * changes (esp. use of builder for serializer construction).
     */
    protected void processViews(SerializationConfig config, BeanSerializerBuilder builder)
    {
        // [JACKSON-232]: whether non-annotated fields are included by default or not is configurable
        List<BeanPropertyWriter> props = builder.getProperties();
        boolean includeByDefault = config.isEnabled(SerializationConfig.Feature.DEFAULT_VIEW_INCLUSION);
        if (includeByDefault) { // non-annotated are included
```

```
        final int propCount = props.size();
        BeanPropertyWriter[] filtered = null;
        // Simple: view information is stored within individual writers, need to combine:
        for (int i = 0; i < propCount; ++i) {
            BeanPropertyWriter bpw = props.get(i);
            Class<?>[] views = bpw.getViews();
            if (views != null) {
                if (filtered == null) {
                    filtered = new BeanPropertyWriter[props.size()];
                }
                filtered[i] = constructFilteredBeanWriter(bpw, views);
            }
        }
        // Anything missing? Need to fill in
        if (filtered != null) {
            for (int i = 0; i < propCount; ++i) {
                if (filtered[i] == null) {
                    filtered[i] = props.get(i);
                }
            }
            builder.setFilteredProperties(filtered);
        }
        // No views, return
        return;
    }
    // Otherwise: only include fields with view definitions
    ArrayList<BeanPropertyWriter> explicit = new ArrayList<BeanPropertyWriter>(props.size());
    for (BeanPropertyWriter bpw : props) {
        Class<?>[] views = bpw.getViews();
        if (views != null) {
            explicit.add(constructFilteredBeanWriter(bpw, views));
        }
    }
    builder.setFilteredProperties(explicit.toArray(new BeanPropertyWriter[explicit.size()]));
}

/**
 * Method that will apply by-type limitations (as per [JACKSON-429]);
 * by default this is based on {@link org.codehaus.jackson.annotate.JsonIgnoreType} annotation but
 * can be supplied by module-provided introspectors too.
 */
protected <T extends AnnotatedMember> void removeIgnorableTypes(SerializationConfig config,
BasicBeanDescription beanDesc,
        Map<String, T> props)
{
    if (props.isEmpty()) {
        return;
    }
```

```
    AnnotationIntrospector intr = config.getAnnotationIntrospector();
    Iterator<Map.Entry<String,T>> it = props.entrySet().iterator();
    HashMap<Class<?>,Boolean> ignores = new HashMap<Class<?>,Boolean>();
    while (it.hasNext()) {
        Map.Entry<String, T> entry = it.next();
        Class<?> type = entry.getValue().getRawType();
        Boolean result = ignores.get(type);
        if (result == null) {
            BasicBeanDescription desc = config.introspectClassAnnotations(type);
            AnnotatedClass ac = desc.getClassInfo();
            result = intr.isIgnorableType(ac);
            // default to false, non-ignorable
            if (result == null) {
                result = Boolean.FALSE;
            }
            ignores.put(type, result);
        }
        // lotsa work, and yes, it is ignorable type, so:
        if (result.booleanValue()) {
            it.remove();
        }
    }
}

/*
/**********************************************************
/* Internal helper methods
/**********************************************************
 */

/**
 * Secondary helper method for constructing {@link BeanPropertyWriter} for
 * given member (field or method).
 */
protected BeanPropertyWriter _constructWriter(SerializationConfig config, TypeBindings typeContext,
        PropertyBuilder pb, boolean staticTyping, String name, AnnotatedMember accessor)
{
    if (config.isEnabled(SerializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS)) {
        accessor.fixAccess();
    }
    JavaType type = accessor.getType(typeContext);
    BeanProperty.Std property = new BeanProperty.Std(name, type, pb.getClassAnnotations(), accessor);

    // Does member specify a serializer? If so, let's use it.
    JsonSerializer<Object> annotatedSerializer = findSerializerFromAnnotation(config, accessor, property);
    // And how about polymorphic typing? First special to cover JAXB per-field settings:
    TypeSerializer contentTypeSer = null;
    if (ClassUtil.isCollectionMapOrArray(type.getRawClass())) {
```

```
            contentTypeSer = findPropertyContentTypeSerializer(type, config, accessor, property);
        }

        // and if not JAXB collection/array with annotations, maybe regular type info?
        TypeSerializer typeSer = findPropertyTypeSerializer(type, config, accessor, property);
        BeanPropertyWriter pbw = pb.buildWriter(name, type, annotatedSerializer,
                    typeSer, contentTypeSer, accessor, staticTyping);
        // how about views? (1.4+)
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
        pbw.setViews(intr.findSerializationViews(accessor));
        return pbw;
    }

    /**
     * Helper method that will sort given List of properties according
     * to defined criteria (usually detected by annotations)
     */
    protected List<BeanPropertyWriter> _sortBeanProperties(List<BeanPropertyWriter> props,
            List<String> creatorProps, String[] propertyOrder, boolean sort)
    {
        int size = props.size();
        Map<String,BeanPropertyWriter> all;
        // Need to (re)sort alphabetically?
        if (sort) {
            all = new TreeMap<String,BeanPropertyWriter>();
        } else {
            all = new LinkedHashMap<String,BeanPropertyWriter>(size*2);
        }

        for (BeanPropertyWriter w : props) {
            all.put(w.getName(), w);
        }
        Map<String,BeanPropertyWriter> ordered = new LinkedHashMap<String,BeanPropertyWriter>(size*2);
        // Ok: primarily by explicit order
        if (propertyOrder != null) {
            for (String name : propertyOrder) {
                BeanPropertyWriter w = all.get(name);
                if (w != null) {
                    ordered.put(name, w);
                }
            }
        }
        // And secondly by sorting Creator properties before other unordered properties
        for (String name : creatorProps) {
            BeanPropertyWriter w = all.get(name);
            if (w != null) {
                ordered.put(name, w);
            }
```

```
        }
        // And finally whatever is left (trying to put again will not change ordering)
        ordered.putAll(all);
        return new ArrayList<BeanPropertyWriter>(ordered.values());
    }
}
package org.codehaus.jackson.map.ser;

import java.lang.reflect.Method;
import java.util.Map;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.introspect.AnnotatedMethod;
import org.codehaus.jackson.map.*;

/**
* Class similar to {@link BeanPropertyWriter}, but that will be used
* for serializing {@link org.codehaus.jackson.annotate.JsonAnyGetter} annotated
* (Map) properties
*
* @since 1.6
*/
public class AnyGetterWriter
{
    protected final Method _anyGetter;

    protected final MapSerializer _serializer;

    public AnyGetterWriter(AnnotatedMethod anyGetter, MapSerializer serializer)
    {
        _anyGetter = anyGetter.getAnnotated();
        _serializer = serializer;
    }

    public void getAndSerialize(Object bean, JsonGenerator jgen, SerializerProvider provider)
        throws Exception
    {
        Object value = _anyGetter.invoke(bean);
        if (value == null) {
            return;
        }
        if (!(value instanceof Map<?,?>)) {
            throw new JsonMappingException("Value returned by 'any-getter' ("+_anyGetter.getName()+"()) not
java.util.Map but "
                    +value.getClass().getName());
        }
        _serializer.serializeFields((Map<?,?>) value, jgen, provider);
    }
```

```
  public void resolve(SerializerProvider provider) throws JsonMappingException
  {
     _serializer.resolve(provider);
  }
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.Type;
import java.text.DateFormat;
import java.util.Date;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.ser.impl.ReadOnlyClassToSerializerMap;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.map.util.RootNameLookup;
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.schema.JsonSchema;
import org.codehaus.jackson.schema.SchemaAware;
import org.codehaus.jackson.type.JavaType;

import org.codehaus.jackson.map.*;

/**
* Default {@link SerializerProvider} implementation. Handles
* caching aspects of serializer handling; all construction details
* are delegated to {@link SerializerFactory} instance.
*<p>
* One note about implementation: the main instance constructed will
* be so-called "blueprint" object, and will NOT be used during actual
* serialization. Rather, an "instance" instance is created so that
* state can be carried along, as well as to avoid synchronization
* during serializer access. Because of this, if sub-classing, one
* must override method {@link #createInstance}: if this is not done,
* an exception will get thrown as base class verifies that the
* instance has same class as the blueprint
* (<code>instance.getClass() == blueprint.getClass()</code>).
* Check is done to prevent weird bugs that would otherwise occur.
*<p>
* Starting with version 1.5, provider is also responsible for
* some parts of type serialization; specifically for locating
* proper type serializers to use for types.
*/
public class StdSerializerProvider
  extends SerializerProvider
{
```

```
  /**
   * Setting for determining whether mappings for "unknown classes" should be
   * cached for faster resolution. Usually this isn't needed, but maybe it
   * is in some cases?
   */
  final static boolean CACHE_UNKNOWN_MAPPINGS = false;

  public final static JsonSerializer<Object> DEFAULT_NULL_KEY_SERIALIZER =
     new FailingSerializer("Null key for a Map not allowed in Json (use a converting NullKeySerializer?)");

  public final static JsonSerializer<Object> DEFAULT_KEY_SERIALIZER = new StdKeySerializer();

  public final static JsonSerializer<Object> DEFAULT_UNKNOWN_SERIALIZER = new
SerializerBase<Object>(Object.class)
  {
     @Override
     public void serialize(Object value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonMappingException
     {
        // 27-Nov-2009, tatu: As per [JACKSON-201] may or may not fail...
        if (provider.isEnabled(SerializationConfig.Feature.FAIL_ON_EMPTY_BEANS)) {
           failForEmpty(value);
        }
        // But if it's fine, we'll just output empty JSON Object:
        jgen.writeStartObject();
        jgen.writeEndObject();
     }

     // since 1.6.2; needed to retain type information
     @Override
     public final void serializeWithType(Object value, JsonGenerator jgen, SerializerProvider provider,
           TypeSerializer typeSer)
        throws IOException, JsonGenerationException
     {
        if (provider.isEnabled(SerializationConfig.Feature.FAIL_ON_EMPTY_BEANS)) {
           failForEmpty(value);
        }
        typeSer.writeTypePrefixForObject(value, jgen);
        typeSer.writeTypeSuffixForObject(value, jgen);
     }

     @Override
     public JsonNode getSchema(SerializerProvider provider, Type typeHint) throws JsonMappingException {
        return null;
     }

     protected void failForEmpty(Object value) throws JsonMappingException
     {
```

```
            throw new JsonMappingException("No serializer found for class "+value.getClass().getName()+" and no properties discovered to create BeanSerializer (to avoid exception, disable SerializationConfig.Feature.FAIL_ON_EMPTY_BEANS) )");
        }
    };

    /*
    /**********************************************************
    /* Configuration, factories
    /**********************************************************
     */

    final protected SerializerFactory _serializerFactory;

    final protected SerializerCache _serializerCache;

    final protected RootNameLookup _rootNames;

    /*
    /**********************************************************
    /* Configuration, specialized serializers
    /**********************************************************
     */

    /**
     * Serializer that gets called for values of types for which no
     * serializers can be constructed.
     *<p>
     * The default serializer will simply thrown an exception; a possible
     * alternative that can be used would be
     * {@link ToStringSerializer}.
     */
    protected JsonSerializer<Object> _unknownTypeSerializer = DEFAULT_UNKNOWN_SERIALIZER;

    /**
     * Serializer used to output non-null keys of Maps (which will get
     * output as Json Objects).
     */
    protected JsonSerializer<Object> _keySerializer = DEFAULT_KEY_SERIALIZER;

    /**
     * Serializer used to output a null value. Default implementation
     * writes nulls using {@link JsonGenerator#writeNull}.
     */
    protected JsonSerializer<Object> _nullValueSerializer = NullSerializer.instance;

    /**
     * Serializer used to (try to) output a null key, due to an entry of
```

```
     * {@link java.util.Map} having null key.
     * The default implementation will throw an exception if this happens;
     * alternative implementation (like one that would write an Empty String)
     * can be defined.
     */
    protected JsonSerializer<Object> _nullKeySerializer = DEFAULT_NULL_KEY_SERIALIZER;

    /*
    /**********************************************************
    /* State, for non-blueprint instances
    /**********************************************************
     */

    /**
     * For fast lookups, we will have a local non-shared read-only
     * map that contains serializers previously fetched.
     */
    protected final ReadOnlyClassToSerializerMap _knownSerializers;

    /**
     * Lazily acquired and instantiated formatter object: initialized
     * first time it is needed, reused afterwards. Used via instances
     * (not blueprints), so that access need not be thread-safe.
     */
    protected DateFormat _dateFormat;

    /*
    /**********************************************************
    /* Life-cycle
    /**********************************************************
     */

    /**
     * Constructor for creating master (or "blue-print") provider object,
     * which is only used as the template for constructing per-binding
     * instances.
     */
    public StdSerializerProvider()
    {
        super(null);
        _serializerFactory = null;
        _serializerCache = new SerializerCache();
        // Blueprints doesn't have access to any serializers...
        _knownSerializers = null;
        _rootNames = new RootNameLookup();
    }

    /**
```

```
 * "Copy-constructor", used from {@link #createInstance} (or by
 * sub-classes)
 *
 * @param src Blueprint object used as the baseline for this instance
 */
protected StdSerializerProvider(SerializationConfig config,
        StdSerializerProvider src, SerializerFactory f)
{
    super(config);
    if (config == null) {
        throw new NullPointerException();
    }
    _serializerFactory = f;

    _serializerCache = src._serializerCache;
    _unknownTypeSerializer = src._unknownTypeSerializer;
    _keySerializer = src._keySerializer;
    _nullValueSerializer = src._nullValueSerializer;
    _nullKeySerializer = src._nullKeySerializer;
    _rootNames = src._rootNames;

    /* Non-blueprint instances do have a read-only map; one that doesn't
     * need synchronization for lookups.
     */
    _knownSerializers = _serializerCache.getReadOnlyLookupMap();
}

/**
 * Overridable method, used to create a non-blueprint instances from the blueprint.
 * This is needed to retain state during serialization.
 */
protected StdSerializerProvider createInstance(SerializationConfig config, SerializerFactory jsf)
{
    return new StdSerializerProvider(config, this, jsf);
}

/*
/**********************************************************
/* Methods to be called by ObjectMapper
/**********************************************************
 */

@Override
public final void serializeValue(SerializationConfig config,
        JsonGenerator jgen, Object value, SerializerFactory jsf)
    throws IOException, JsonGenerationException
{
    if (jsf == null) {
```

```
            throw new IllegalArgumentException("Can not pass null serializerFactory");
        }

        /* First: we need a separate instance, which will hold a copy of the
         * non-shared ("local") read-only lookup Map for fast
         * class-to-serializer lookup
         */
        StdSerializerProvider inst = createInstance(config, jsf);
        // sanity check to avoid weird errors; to ensure sub-classes do override createInstance
        if (inst.getClass() != getClass()) {
            throw new IllegalStateException("Broken serializer provider: createInstance returned instance of type "+inst.getClass()+"; blueprint of type "+getClass());
        }
        // And then we can do actual serialization, through the instance
        inst._serializeValue(jgen, value);
    }

    @Override
    public final void serializeValue(SerializationConfig config, JsonGenerator jgen,
            Object value, JavaType rootType, SerializerFactory jsf)
        throws IOException, JsonGenerationException
    {
        if (jsf == null) {
            throw new IllegalArgumentException("Can not pass null serializerFactory");
        }
        StdSerializerProvider inst = createInstance(config, jsf);
        if (inst.getClass() != getClass()) {
            throw new IllegalStateException("Broken serializer provider: createInstance returned instance of type "+inst.getClass()+"; blueprint of type "+getClass());
        }
        inst._serializeValue(jgen, value, rootType);
    }

    @Override
    public JsonSchema generateJsonSchema(Class<?> type, SerializationConfig config, SerializerFactory jsf)
            throws JsonMappingException
    {
        if (type == null) {
            throw new IllegalArgumentException("A class must be provided");
        }

        /* First: we need a separate instance, which will hold a copy of the
         * non-shared ("local") read-only lookup Map for fast
         * class-to-serializer lookup
         */
        StdSerializerProvider inst = createInstance(config, jsf);
        // sanity check to avoid weird errors; to ensure sub-classes do override createInstance
        if (inst.getClass() != getClass()) {
```

```
            throw new IllegalStateException("Broken serializer provider: createInstance returned instance of type
"+inst.getClass()+"; blueprint of type "+getClass());
        }
        /* no need for embedded type information for JSON schema generation (all
         * type information it needs is accessible via "untyped" serializer)
         */
        JsonSerializer<Object> ser = inst.findValueSerializer(type, null);
        JsonNode schemaNode = (ser instanceof SchemaAware) ?
                ((SchemaAware) ser).getSchema(inst, null) :
                JsonSchema.getDefaultSchemaNode();
        if (!(schemaNode instanceof ObjectNode)) {
            throw new IllegalArgumentException("Class " + type.getName() +
                    " would not be serialized as a JSON object and therefore has no schema");
        }

        return new JsonSchema((ObjectNode) schemaNode);
    }

    @Override
    public boolean hasSerializerFor(SerializationConfig config,
                                    Class<?> cls, SerializerFactory jsf)
    {
        return createInstance(config, jsf)._findExplicitUntypedSerializer(cls, null) != null;
    }

    /*
    /**********************************************************
    /* Configuration methods
    /**********************************************************
     */

    public void setKeySerializer(JsonSerializer<Object> ks)
    {
        if (ks == null) {
            throw new IllegalArgumentException("Can not pass null JsonSerializer");
        }
        _keySerializer = ks;
    }

    public void setNullValueSerializer(JsonSerializer<Object> nvs)
    {
        if (nvs == null) {
            throw new IllegalArgumentException("Can not pass null JsonSerializer");
        }
        _nullValueSerializer = nvs;
    }

    public void setNullKeySerializer(JsonSerializer<Object> nks)
```

```
{
   if (nks == null) {
      throw new IllegalArgumentException("Can not pass null JsonSerializer");
   }
   _nullKeySerializer = nks;
}

@Override
public int cachedSerializersCount() {
   return _serializerCache.size();
}

@Override
public void flushCachedSerializers() {
   _serializerCache.flush();
}

/*
/**********************************************************
/* Abstract method implementations, value/type serializers
/**********************************************************
 */

@Override
@SuppressWarnings("unchecked")
public JsonSerializer<Object> findValueSerializer(Class<?> valueType,
      BeanProperty property)
   throws JsonMappingException
{
   // Fast lookup from local lookup thingy works?
   JsonSerializer<Object> ser = _knownSerializers.untypedValueSerializer(valueType);
   if (ser == null) {
      // If not, maybe shared map already has it?
      ser = _serializerCache.untypedValueSerializer(valueType);
      if (ser == null) {
         // ... possibly as fully typed?
         ser = _serializerCache.untypedValueSerializer(TypeFactory.type(valueType));
         if (ser == null) {
            // If neither, must create
            ser = _createAndCacheUntypedSerializer(valueType, property);
            // Not found? Must use the unknown type serializer
            /* Couldn't create? Need to return the fallback serializer, which
             * most likely will report an error: but one question is whether
             * we should cache it?
             */
            if (ser == null) {
               ser = getUnknownTypeSerializer(valueType);
               // Should this be added to lookups?
```

```
                if (CACHE_UNKNOWN_MAPPINGS) {
                    _serializerCache.addNonTypedSerializer(valueType, ser);
                }
                return ser;
            }
        }
    }
    if (ser instanceof ContextualSerializer<?>) {
        return ((ContextualSerializer<Object>) ser).createContextual(_config, property);
    }
    return ser;
}

/**
 * This variant was added in 1.5, to allow for efficient access using full
 * structured types, not just classes. This is necessary for accurate
 * handling of external type information, to handle polymorphic types.
 */
@SuppressWarnings("unchecked")
@Override
public JsonSerializer<Object> findValueSerializer(JavaType valueType, BeanProperty property)
    throws JsonMappingException
{
    // Fast lookup from local lookup thingy works?
    JsonSerializer<Object> ser = _knownSerializers.untypedValueSerializer(valueType);
    if (ser == null) {
        // If not, maybe shared map already has it?
        ser = _serializerCache.untypedValueSerializer(valueType);
        if (ser == null) {
            // If neither, must create
            ser = _createAndCacheUntypedSerializer(valueType, property);
            // Not found? Must use the unknown type serializer
            /* Couldn't create? Need to return the fallback serializer, which
             * most likely will report an error: but one question is whether
             * we should cache it?
             */
            if (ser == null) {
                ser = getUnknownTypeSerializer(valueType.getRawClass());
                // Should this be added to lookups?
                if (CACHE_UNKNOWN_MAPPINGS) {
                    _serializerCache.addNonTypedSerializer(valueType, ser);
                }
                return ser;
            }
        }
    }
    if (ser instanceof ContextualSerializer<?>) {
```

```
            return ((ContextualSerializer<Object>) ser).createContextual(_config, property);
        }
        return ser;
    }

    /**
     * @param cache Whether resulting value serializer should be cached or not; this is just
     *    a hint
     */
    @Override
    public JsonSerializer<Object> findTypedValueSerializer(Class<?> valueType, boolean cache,
            BeanProperty property)
        throws JsonMappingException
    {
        // Two-phase lookups; local non-shared cache, then shared:
        JsonSerializer<Object> ser = _knownSerializers.typedValueSerializer(valueType);
        if (ser != null) {
            return ser;
        }
        // If not, maybe shared map already has it?
        ser = _serializerCache.typedValueSerializer(valueType);
        if (ser != null) {
            return ser;
        }

        // Well, let's just compose from pieces:
        ser = findValueSerializer(valueType, property);
        TypeSerializer typeSer = _serializerFactory.createTypeSerializer(_config,
                TypeFactory.type(valueType), property);
        if (typeSer != null) {
            ser = new WrappedSerializer(typeSer, ser);
        }
        if (cache) {
            _serializerCache.addTypedSerializer(valueType, ser);
        }
        return ser;
    }

    @Override
    public JsonSerializer<Object> findTypedValueSerializer(JavaType valueType, boolean cache,
            BeanProperty property)
        throws JsonMappingException
    {
        // Two-phase lookups; local non-shared cache, then shared:
        JsonSerializer<Object> ser = _knownSerializers.typedValueSerializer(valueType);
        if (ser != null) {
            return ser;
        }
```

```
    // If not, maybe shared map already has it?
    ser = _serializerCache.typedValueSerializer(valueType);
    if (ser != null) {
        return ser;
    }

    // Well, let's just compose from pieces:
    ser = findValueSerializer(valueType, property);
    TypeSerializer typeSer = _serializerFactory.createTypeSerializer(_config, valueType, property);
    if (typeSer != null) {
        ser = new WrappedSerializer(typeSer, ser);
    }
    if (cache) {
        _serializerCache.addTypedSerializer(valueType, ser);
    }
    return ser;
}

/*
/**********************************************************
/* Abstract method implementations, other serializers
/**********************************************************
 */

@Override
public JsonSerializer<Object> getKeySerializer(JavaType valueType, BeanProperty property)
{
    return _keySerializer;
}

@Override
public JsonSerializer<Object> getNullKeySerializer() {
    return _nullKeySerializer;
}

@Override
public JsonSerializer<Object> getNullValueSerializer() {
    return _nullValueSerializer;
}

@Override
public JsonSerializer<Object> getUnknownTypeSerializer(Class<?> unknownType) {
    return _unknownTypeSerializer;
}

/*
/**********************************************************
/* Abstract method impls, convenience methods
```

```
/**********************************************************
 */

/**
 * @param timestamp Millisecond timestamp that defines date, if available;
 */
@Override
public final void defaultSerializeDateValue(long timestamp, JsonGenerator jgen)
   throws IOException, JsonProcessingException
{
   // [JACKSON-87]: Support both numeric timestamps and textual
   if (isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)) {
      jgen.writeNumber(timestamp);
   } else {
      if (_dateFormat == null) {
         // must create a clone since Formats are not thread-safe:
         _dateFormat = (DateFormat)_config.getDateFormat().clone();
      }
      jgen.writeString(_dateFormat.format(new Date(timestamp)));
   }
}

@Override
public final void defaultSerializeDateValue(Date date, JsonGenerator jgen)
   throws IOException, JsonProcessingException
{
   // [JACKSON-87]: Support both numeric timestamps and textual
   if (isEnabled(SerializationConfig.Feature.WRITE_DATES_AS_TIMESTAMPS)) {
      jgen.writeNumber(date.getTime());
   } else {
      if (_dateFormat == null) {
         DateFormat blueprint = _config.getDateFormat();
         // must create a clone since Formats are not thread-safe:
         _dateFormat = (DateFormat)blueprint.clone();
      }
      jgen.writeString(_dateFormat.format(date));
   }
}

/*
/**********************************************************
/* Helper methods: can be overridden by sub-classes
/**********************************************************
 */

/**
 * Method called on the actual non-blueprint provider instance object,
 * to kick off the serialization.
```

```
 */
protected  void _serializeValue(JsonGenerator jgen, Object value)
   throws IOException, JsonProcessingException
{
   JsonSerializer<Object> ser;
   boolean wrap;

   if (value == null) {
      ser = getNullValueSerializer();
      wrap = false; // no name to use for wrapping; can't do!
   } else {
      Class<?> cls = value.getClass();
      // true, since we do want to cache root-level typed serializers (ditto for null property)
      ser = findTypedValueSerializer(cls, true, null);
      // [JACKSON-163]
      wrap = _config.isEnabled(SerializationConfig.Feature.WRAP_ROOT_VALUE);
      if (wrap) {
         jgen.writeStartObject();
         jgen.writeFieldName(_rootNames.findRootName(value.getClass(), _config));
      }
   }
   try {
      ser.serialize(value, jgen, this);
      if (wrap) {
         jgen.writeEndObject();
      }
   } catch (IOException ioe) {
      /* As per [JACKSON-99], should not wrap IOException or its
       * sub-classes (like JsonProcessingException, JsonMappingException)
       */
      throw ioe;
   } catch (Exception e) {
      // but others are wrapped
      String msg = e.getMessage();
      if (msg == null) {
         msg = "[no message for "+e.getClass().getName()+"]";
      }
      throw new JsonMappingException(msg, e);
   }
}

/**
 * Method called on the actual non-blueprint provider instance object,
 * to kick off the serialization, when root type is explicitly
 * specified and not determined from value.
 */
protected  void _serializeValue(JsonGenerator jgen, Object value, JavaType rootType)
   throws IOException, JsonProcessingException
```

```
{
    // [JACKSON-163]
    boolean wrap;

    JsonSerializer<Object> ser;
    if (value == null) {
        ser = getNullValueSerializer();
        wrap = false;
    } else {
        // Let's ensure types are compatible at this point
        if (!rootType.getRawClass().isAssignableFrom(value.getClass())) {
            _reportIncompatibleRootType(value, rootType);
        }
        // root value, not reached via property:
        ser = findTypedValueSerializer(rootType, true, null);
        // [JACKSON-163]
        wrap = _config.isEnabled(SerializationConfig.Feature.WRAP_ROOT_VALUE);
        if (wrap) {
            jgen.writeStartObject();
            jgen.writeFieldName(_rootNames.findRootName(rootType, _config));
        }
    }
    try {
        ser.serialize(value, jgen, this);
        if (wrap) {
            jgen.writeEndObject();
        }
    } catch (IOException ioe) { // no wrapping for IO (and derived)
        throw ioe;
    } catch (Exception e) { // but others do need to be, to get path etc
        String msg = e.getMessage();
        if (msg == null) {
            msg = "[no message for "+e.getClass().getName()+"]";
        }
        throw new JsonMappingException(msg, e);
    }
}

protected void _reportIncompatibleRootType(Object value, JavaType rootType)
    throws IOException, JsonProcessingException
{
    /* 07-Jan-2010, tatu: As per [JACKSON-456] better handle distinction between wrapper types,
     *    primitives
     */
    if (rootType.isPrimitive()) {
        Class<?> wrapperType = ClassUtil.wrapperType(rootType.getRawClass());
        // If it's just difference between wrapper, primitive, let it slide
        if (wrapperType.isAssignableFrom(value.getClass())) {
```

```
            return;
        }
    }
    throw new JsonMappingException("Incompatible types: declared root type ("+rootType+") vs "
            +value.getClass().getName());
}

/**
 * Method that will try to find a serializer, either from cache
 * or by constructing one; but will not return an "unknown" serializer
 * if this can not be done but rather returns null.
 *
 * @return Serializer if one can be found, null if not.
 */
protected JsonSerializer<Object> _findExplicitUntypedSerializer(Class<?> runtimeType,
        BeanProperty property)
{
    // Fast lookup from local lookup thingy works?
    JsonSerializer<Object> ser = _knownSerializers.untypedValueSerializer(runtimeType);
    if (ser != null) {
        return ser;
    }
    // If not, maybe shared map already has it?
    ser = _serializerCache.untypedValueSerializer(runtimeType);
    if (ser != null) {
        return ser;
    }
    try {
        return _createAndCacheUntypedSerializer(runtimeType, property);
    } catch (Exception e) {
        return null;
    }
}

/**
 * Method that will try to construct a value serializer; and if
 * one is succesfully created, cache it for reuse.
 */
protected JsonSerializer<Object> _createAndCacheUntypedSerializer(Class<?> type,
        BeanProperty property)
    throws JsonMappingException
{
    JsonSerializer<Object> ser;
    try {
        ser = _createUntypedSerializer(TypeFactory.type(type), property);
    } catch (IllegalArgumentException iae) {
        /* We better only expose checked exceptions, since those
         * are what caller is expected to handle
```

```
             */
            throw new JsonMappingException(iae.getMessage(), null, iae);
        }

        if (ser != null) {
            _serializerCache.addNonTypedSerializer(type, ser);
            /* Finally: some serializers want to do post-processing, after
             * getting registered (to handle cyclic deps).
             */
            if (ser instanceof ResolvableSerializer) {
                _resolveSerializer((ResolvableSerializer)ser);
            }
        }
        return ser;
    }

    /**
     * @since 1.5
]    */
    protected JsonSerializer<Object> _createAndCacheUntypedSerializer(JavaType type,
            BeanProperty property)
        throws JsonMappingException
    {
        JsonSerializer<Object> ser;
        try {
            ser = _createUntypedSerializer(type, property);
        } catch (IllegalArgumentException iae) {
            /* We better only expose checked exceptions, since those
             * are what caller is expected to handle
             */
            throw new JsonMappingException(iae.getMessage(), null, iae);
        }

        if (ser != null) {
            _serializerCache.addNonTypedSerializer(type, ser);
            /* Finally: some serializers want to do post-processing, after
             * getting registered (to handle cyclic deps).
             */
            if (ser instanceof ResolvableSerializer) {
                _resolveSerializer((ResolvableSerializer)ser);
            }
        }
        return ser;
    }

    protected JsonSerializer<Object> _createUntypedSerializer(JavaType type,
            BeanProperty property)
        throws JsonMappingException
```

```
{
    /* 10-Dec-2008, tatu: Is there a possibility of infinite loops
     *   here? Shouldn't be, given that we do not pass back-reference
     *   to this provider. But if there is, we'd need to sync calls,
     *   and keep track of creation chain to look for loops -- fairly
     *   easy to do, but won't add yet since it seems unnecessary.
     */
    return (JsonSerializer<Object>)_serializerFactory.createSerializer(_config, type, property);
}

protected void _resolveSerializer(ResolvableSerializer ser)
    throws JsonMappingException
{
    ser.resolve(this);
}

/*
/**********************************************************
/*  Helper classes
/**********************************************************
 */

/**
 * Simple serializer that will call configured type serializer, passing
 * in configured data serializer, and exposing it all as a simple
 * serializer.
 */
private final static class WrappedSerializer
    extends JsonSerializer<Object>
{
    final protected TypeSerializer _typeSerializer;
    final protected JsonSerializer<Object> _serializer;

    public WrappedSerializer(TypeSerializer typeSer, JsonSerializer<Object> ser)
    {
        super();
        _typeSerializer = typeSer;
        _serializer = ser;
    }

    @Override
    public void serialize(Object value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonProcessingException
    {
        _serializer.serializeWithType(value, jgen, provider, _typeSerializer);
    }

    @Override
```

```
        public void serializeWithType(Object value, JsonGenerator jgen, SerializerProvider provider,
                TypeSerializer typeSer)
            throws IOException, JsonProcessingException
        {
            /* Is this an erroneous call? For now, let's assume it is not, and
             * that type serializer is just overridden if so
             */
            _serializer.serializeWithType(value, jgen, provider, typeSer);
        }

        @Override
        public Class<Object> handledType() { return Object.class; }
    }
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
import java.lang.reflect.Type;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.schema.SchemaAware;
import org.codehaus.jackson.schema.JsonSchema;
import org.codehaus.jackson.type.JavaType;

/**
* Serializer class that can serialize Object that have a
* {@link org.codehaus.jackson.annotate.JsonValue} annotation to
* indicate that serialization should be done by calling the method
* annotated, and serializing result it returns.
* <p/>
* Implementation note: we will post-process resulting serializer
* (much like what is done with {@link BeanSerializer})
* to figure out actual serializers for final types. This must be
* done from {@link #resolve} method, and NOT from constructor;
* otherwise we could end up with an infinite loop.
*/
@JacksonStdImpl
public final class JsonValueSerializer
    extends SerializerBase<Object>
```

```
  implements ResolvableSerializer, SchemaAware
{
  protected final Method _accessorMethod;

  protected JsonSerializer<Object> _valueSerializer;

  protected final BeanProperty _property;

  /**
   * This is a flag that is set in rare (?) cases where this serializer
   * is used for "natural" types (boolean, int, String, double); and where
   * we actually must force type information wrapping, even though
   * one would not normally be added.
   *
   * @since 1.7
   */
  protected boolean _forceTypeInformation;

  /**
   * @param ser Explicit serializer to use, if caller knows it (which
   *            occurs if and only if the "value method" was annotated with
   *            {@link org.codehaus.jackson.map.annotate.JsonSerialize#using}), otherwise
   *            null
   */
  public JsonValueSerializer(Method valueMethod, JsonSerializer<Object> ser, BeanProperty property)
  {
     super(Object.class);
     _accessorMethod = valueMethod;
     _valueSerializer = ser;
     _property = property;
  }

  @Override
  public void serialize(Object bean, JsonGenerator jgen, SerializerProvider prov)
     throws IOException, JsonGenerationException
  {
     try {
        Object value = _accessorMethod.invoke(bean);

        if (value == null) {
           prov.defaultSerializeNull(jgen);
           return;
        }
        JsonSerializer<Object> ser = _valueSerializer;
        if (ser == null) {
           Class<?> c = value.getClass();
           /* 10-Mar-2010, tatu: Ideally we would actually separate out type
            *   serializer from value serializer; but, alas, there's no access
```

```
             *   to serializer factory at this point...
             */
            // let's cache it, may be needed soon again
            ser = prov.findTypedValueSerializer(c, true, _property);
        }
        ser.serialize(value, jgen, prov);
    } catch (IOException ioe) {
        throw ioe;
    } catch (Exception e) {
        Throwable t = e;
        // Need to unwrap this specific type, to see infinite recursion...
        while (t instanceof InvocationTargetException && t.getCause() != null) {
            t = t.getCause();
        }
        // Errors shouldn't be wrapped (and often can't, as well)
        if (t instanceof Error) {
            throw (Error) t;
        }
        // let's try to indicate the path best we can...
        throw JsonMappingException.wrapWithPath(t, bean, _accessorMethod.getName() + "()");
    }
}

@Override
public void serializeWithType(Object bean, JsonGenerator jgen, SerializerProvider provider,
        TypeSerializer typeSer)
    throws IOException, JsonProcessingException
{
    // Regardless of other parts, first need to find value to serialize:
    Object value = null;
    try {
        value = _accessorMethod.invoke(bean);

        // and if we got null, can also just write it directly
        if (value == null) {
            provider.defaultSerializeNull(jgen);
            return;
        }
        JsonSerializer<Object> ser = _valueSerializer;
        if (ser != null) { // already got a serializer? fabulous, that be easy...
            /* 09-Dec-2010, tatu: To work around natural type's refusal to add type info, we do
             *    this (note: type is for the wrapper type, not enclosed value!)
             */
            if (_forceTypeInformation) {
                typeSer.writeTypePrefixForScalar(bean, jgen);
            }
            ser.serializeWithType(value, jgen, provider, typeSer);
            if (_forceTypeInformation) {
```

```
                typeSer.writeTypeSuffixForScalar(bean, jgen);
            }
            return;
        }
        // But if not, it gets tad trickier (copied from main serialize() method)
        Class<?> c = value.getClass();
        ser = provider.findTypedValueSerializer(c, true, _property);
        // note: now we have bundled type serializer, so should NOT call with typed version
        ser.serialize(value, jgen, provider);
    } catch (IOException ioe) {
        throw ioe;
    } catch (Exception e) {
        Throwable t = e;
        // Need to unwrap this specific type, to see infinite recursion...
        while (t instanceof InvocationTargetException && t.getCause() != null) {
            t = t.getCause();
        }
        // Errors shouldn't be wrapped (and often can't, as well)
        if (t instanceof Error) {
            throw (Error) t;
        }
        // let's try to indicate the path best we can...
        throw JsonMappingException.wrapWithPath(t, bean, _accessorMethod.getName() + "()");
    }
}

@Override
public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    throws JsonMappingException
{
    return (_valueSerializer instanceof SchemaAware) ?
            ((SchemaAware) _valueSerializer).getSchema(provider, null) :
            JsonSchema.getDefaultSchemaNode();
}

/*
/**********************************************************
/* ResolvableSerializer impl
/**********************************************************
 */

/**
 * We can try to find the actual serializer for value, if we can
 * statically figure out what the result type must be.
 */
public void resolve(SerializerProvider provider)
    throws JsonMappingException
{
```

```
    if (_valueSerializer == null) {
        /* Note: we can only assign serializer statically if the
         * declared type is final -- if not, we don't really know
         * the actual type until we get the instance.
         */
        // 10-Mar-2010, tatu: Except if static typing is to be used
        if (provider.isEnabled(SerializationConfig.Feature.USE_STATIC_TYPING)
                || Modifier.isFinal(_accessorMethod.getReturnType().getModifiers())) {
            JavaType t = TypeFactory.type(_accessorMethod.getGenericReturnType());
            // false -> no need to cache
            /* 10-Mar-2010, tatu: Ideally we would actually separate out type
             *   serializer from value serializer; but, alas, there's no access
             *   to serializer factory at this point...
             */
            _valueSerializer = provider.findTypedValueSerializer(t, false, _property);
            /* 09-Dec-2010, tatu: Turns out we must add special handling for
             *   cases where "native" (aka "natural") type is being serialized,
             *   using standard serializer
             */
            _forceTypeInformation = isNaturalTypeWithStdHandling(t, _valueSerializer);
        }
    }
}

protected boolean isNaturalTypeWithStdHandling(JavaType type, JsonSerializer<?> ser)
{
    Class<?> cls = type.getRawClass();
    // First: do we have a natural type being handled?
    if (type.isPrimitive()) {
        if (cls != Integer.TYPE && cls != Boolean.TYPE && cls != Double.TYPE) {
            return false;
        }
    } else {
        if (cls != String.class &&
                cls != Integer.class && cls != Boolean.class && cls != Double.class) {
            return false;
        }
    }
    // Second: and it's handled with standard serializer?
    return (ser.getClass().getAnnotation(JacksonStdImpl.class)) != null;
}

/*
/**********************************************************
/* Other methods
/**********************************************************
 */
```

```
  @Override
  public String toString()
  {
     return "(@JsonValue serializer for method " + _accessorMethod.getDeclaringClass() + "#" +
_accessorMethod.getName() + ")";
  }
}
package org.codehaus.jackson.map.ser;

import java.util.*;

import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.introspect.BasicBeanDescription;

/**
* Builder class used for aggregating deserialization information about
* a POJO, in order to build a {@link JsonSerializer} for serializing
* intances.
* Main reason for using separate builder class is that this makes it easier
* to make actual serializer class fully immutable.
*
* @since 1.7
*/
public class BeanSerializerBuilder
{
  private final static BeanPropertyWriter[] NO_PROPERTIES = new BeanPropertyWriter[0];

  /*
  /**********************************************************
  /* General information about POJO
  /**********************************************************
   */

  final protected BasicBeanDescription _beanDesc;

  /*
  /**********************************************************
  /* Accumulated information about properties
  /**********************************************************
   */

  /**
   * Bean properties, in order of serialization
   */
  protected List<BeanPropertyWriter> _properties;

  /**
   * Optional array of filtered property writers; if null, no
```

```
     * view-based filtering is performed.
     */
    protected BeanPropertyWriter[] _filteredProperties;

    /**
     * Writer used for "any getter" properties, if any.
     */
    protected AnyGetterWriter _anyGetter;

    /**
     * Id of the property filter to use for POJO, if any.
     */
    protected Object _filterId;

    /*
    /**********************************************************
    /* Construction and setter methods
    /**********************************************************
     */

    public BeanSerializerBuilder(BasicBeanDescription beanDesc) {
        _beanDesc = beanDesc;
    }

    /**
     * Copy-constructor that may be used for sub-classing
     */
    protected BeanSerializerBuilder(BeanSerializerBuilder src) {
        _beanDesc = src._beanDesc;
        _properties = src._properties;
        _filteredProperties = src._filteredProperties;
        _anyGetter = src._anyGetter;
        _filterId = src._filterId;
    }

    public BasicBeanDescription getBeanDescription() { return _beanDesc; }
    public List<BeanPropertyWriter> getProperties() { return _properties; }
    public BeanPropertyWriter[] getFilteredProperties() { return _filteredProperties; }

    public void setProperties(List<BeanPropertyWriter> properties) {
        _properties = properties;
    }

    public void setFilteredProperties(BeanPropertyWriter[] properties) {
        _filteredProperties = properties;
    }

    public void setAnyGetter(AnyGetterWriter anyGetter) {
```

```
        _anyGetter = anyGetter;
    }

    public void setFilterId(Object filterId) {
        _filterId = filterId;
    }

    /*
    /**********************************************************
    /* Build methods for actually creating serializer instance
    /**********************************************************
     */

    /**
     * Method called to create {@link BeanSerializer} instance with
     * all accumulated information.
     */
    public JsonSerializer<?> build()
    {
        BeanPropertyWriter[] properties = (_properties == null || _properties.isEmpty()) ?
                NO_PROPERTIES : _properties.toArray(new BeanPropertyWriter[_properties.size()]);
        return new BeanSerializer(_beanDesc.getType(), properties, _filteredProperties,
                _anyGetter, _filterId);
    }

    /**
     * Factory method for constructing an "empty" serializer; one that
     * outputs no properties (but handles JSON objects properly, including
     * type information)
     */
    public BeanSerializer createDummy() {
        return BeanSerializer.createDummy(_beanDesc.getBeanClass());
    }
}
package org.codehaus.jackson.map.ser;

import java.util.*;

import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.ser.impl.ReadOnlyClassToSerializerMap;

/**
* Simple cache object that allows for doing 2-level lookups: first level is
* by "local" read-only lookup Map (used without locking)
* and second backup level is by a shared modifiable HashMap.
* The idea is that after a while, most serializers are found from the
* local Map (to optimize performance, reduce lock contention),
```

```
* but that during buildup we can use a shared map to reduce both
* number of distinct read-only maps constructed, and number of
* serializers constructed.
*<p>
* Since version 1.5 cache will actually contain three kinds of entries,
* based on combination of class pair key. First class in key is for the
* type to serialize, and second one is type used for determining how
* to resolve value type. One (but not both) of entries can be null.
*/
public final class SerializerCache
{
  /**
   * Shared, modifiable map; all access needs to be through synchronized blocks.
   *<p>
   * NOTE: keys are of various types (see below for key types), in addition to
   * basic {@link JavaType} used for "untyped" serializers.
   */
  private HashMap<TypeKey, JsonSerializer<Object>> _sharedMap = new HashMap<TypeKey,
JsonSerializer<Object>>(64);

  /**
   * Most recent read-only instance, created from _sharedMap, if any.
   */
  private ReadOnlyClassToSerializerMap _readOnlyMap = null;

  public SerializerCache() {
  }

  /**
   * Method that can be called to get a read-only instance populated from the
   * most recent version of the shared lookup Map.
   */
  public ReadOnlyClassToSerializerMap getReadOnlyLookupMap()
  {
     ReadOnlyClassToSerializerMap m;
     synchronized (this) {
        m = _readOnlyMap;
        if (m == null) {
           _readOnlyMap = m = ReadOnlyClassToSerializerMap.from(_sharedMap);
        }
     }
     return m.instance();
  }

  /**
   * Method that checks if the shared (and hence, synchronized) lookup Map might have
   * untyped serializer for given type.
   */
```

```
public JsonSerializer<Object> untypedValueSerializer(Class<?> type)
{
   synchronized (this) {
      return _sharedMap.get(new TypeKey(type, false));
   }
}

/**
 * @since 1.5
 */
public JsonSerializer<Object> untypedValueSerializer(JavaType type)
{
   synchronized (this) {
      return _sharedMap.get(new TypeKey(type, false));
   }
}

public JsonSerializer<Object> typedValueSerializer(JavaType type)
{
   synchronized (this) {
      return _sharedMap.get(new TypeKey(type, true));
   }
}

public JsonSerializer<Object> typedValueSerializer(Class<?> cls)
{
   synchronized (this) {
      return _sharedMap.get(new TypeKey(cls, true));
   }
}

/**
 * Method called if none of lookups succeeded, and caller had to construct
 * a serializer. If so, we will update the shared lookup map so that it
 * can be resolved via it next time.
 */
public void addTypedSerializer(JavaType type, JsonSerializer<Object> ser)
{
   synchronized (this) {
      if (_sharedMap.put(new TypeKey(type, true), ser) == null) {
         // let's invalidate the read-only copy, too, to get it updated
         _readOnlyMap = null;
      }
   }
}

public void addTypedSerializer(Class<?> cls, JsonSerializer<Object> ser)
{
```

```
    synchronized (this) {
        if (_sharedMap.put(new TypeKey(cls, true), ser) == null) {
            // let's invalidate the read-only copy, too, to get it updated
            _readOnlyMap = null;
        }
    }
}

public void addNonTypedSerializer(Class<?> type, JsonSerializer<Object> ser)
{
    synchronized (this) {
        if (_sharedMap.put(new TypeKey(type, false), ser) == null) {
            // let's invalidate the read-only copy, too, to get it updated
            _readOnlyMap = null;
        }
    }
}

/**
 * @since 1.5
 */
public void addNonTypedSerializer(JavaType type, JsonSerializer<Object> ser)
{
    synchronized (this) {
        if (_sharedMap.put(new TypeKey(type, false), ser) == null) {
            // let's invalidate the read-only copy, too, to get it updated
            _readOnlyMap = null;
        }
    }
}

/**
 * @since 1.4
 */
public synchronized int size() {
    return _sharedMap.size();
}

public synchronized void flush() {
    _sharedMap.clear();
}

/*
/**********************************************************************
/* Helper class(es)
/**********************************************************************
 */
```

```
/**
 * Key that offers two "modes"; one with raw class, as used for
 * cases were raw class type is available (for example, when using
 * runtime type); and one with full generics-including.
 */
public final static class TypeKey
{
    protected int _hashCode;

    protected Class<?> _class;

    protected JavaType _type;

    /**
     * Indicator of whether serializer stored has a type serializer
     * wrapper around it or not; if not, it is "untyped" serializer;
     * if it has, it is "typed"
     */
    protected boolean _isTyped;

    public TypeKey(Class<?> key, boolean typed) {
        _class = key;
        _type = null;
        _isTyped = typed;
        _hashCode = hash(key, typed);
    }

    public TypeKey(JavaType key, boolean typed) {
        _type = key;
        _class = null;
        _isTyped = typed;
        _hashCode = hash(key, typed);
    }

    private final static int hash(Class<?> cls, boolean typed) {
        int hash = cls.getName().hashCode();
        if (typed) {
            ++hash;
        }
        return hash;
    }

    private final static int hash(JavaType type, boolean typed) {
        int hash = type.hashCode() - 1;
        if (typed) {
            --hash;
        }
        return hash;
```

```
}

public void resetTyped(Class<?> cls) {
   _type = null;
   _class = cls;
   _isTyped = true;
   _hashCode = hash(cls, true);
}

public void resetUntyped(Class<?> cls) {
   _type = null;
   _class = cls;
   _isTyped = false;
   _hashCode = hash(cls, false);
}

public void resetTyped(JavaType type) {
   _type = type;
   _class = null;
   _isTyped = true;
   _hashCode = hash(type, true);
}

public void resetUntyped(JavaType type) {
   _type = type;
   _class = null;
   _isTyped = false;
   _hashCode = hash(type, false);
}

@Override public final int hashCode() { return _hashCode; }

@Override public final String toString() {
   if (_class != null) {
      return "{class: "+_class.getName()+", typed? "+_isTyped+"}";
   }
   return "{type: "+_type+", typed? "+_isTyped+"}";
}

// note: we assume key is never used for anything other than as map key, so:
@Override public final boolean equals(Object o)
{
   if (o == this) return true;
   TypeKey other = (TypeKey) o;
   if (other._isTyped == _isTyped) {
      if (_class != null) {
         return other._class == _class;
      }
```

```
            return _type.equals(other._type);
        }
        return false;
    }
  }
}
package org.codehaus.jackson.map.ser;

/**
* Interface for objects that providers instances of {@link BeanPropertyFilter}
* that match given ids. A provider is configured to be used during serialization,
* to find filter to used based on id specified by {@link org.codehaus.jackson.map.annotate.JsonFilter}
* annotation on bean class.
*
* @since 1.7
*/
public abstract class FilterProvider
{
  /**
   * Lookup method used to find {@link BeanPropertyFilter} that has specified id.
   * Note that id is typically a {@link java.lang.String}, but is not necessarily
   * limited to that; that is, while standard components use String, custom
   * implementation can choose other kinds of keys.
   *
   * @return Filter registered with specified id, if one defined; null if
   *   none found.
   */
  public abstract BeanPropertyFilter findFilter(Object filterId);
}
package org.codehaus.jackson.map.ser;

import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Decorated {@link BeanPropertyWriter} that will filter out
* properties that are not to be included in currently active
* JsonView.
*
* @since 1.4
*/
public abstract class FilteredBeanPropertyWriter
{
  public static BeanPropertyWriter constructViewBased(BeanPropertyWriter base, Class<?>[] viewsToIncludeIn)
  {
    if (viewsToIncludeIn.length == 1) {
        return new SingleView(base, viewsToIncludeIn[0]);
```

```
    }
    return new MultiView(base, viewsToIncludeIn);
}

/*
/**********************************************************
/* Concrete sub-classes
/**********************************************************
 */

private final static class SingleView
    extends BeanPropertyWriter
{
    protected final Class<?> _view;

    protected SingleView(BeanPropertyWriter base, Class<?> view) {
        super(base);
        _view = view;
    }

    protected SingleView(SingleView fromView, JsonSerializer<Object> ser) {
        super(fromView, ser);
        _view = fromView._view;
    }

    @Override
    public BeanPropertyWriter withSerializer(JsonSerializer<Object> ser)
    {
        return new SingleView(this, ser);
    }

    @Override
    public void serializeAsField(Object bean, JsonGenerator jgen, SerializerProvider prov)
        throws Exception
    {
        Class<?> activeView = prov.getSerializationView();
        if (activeView == null || _view.isAssignableFrom(activeView)) {
            super.serializeAsField(bean, jgen, prov);
        }
    }
}

private final static class MultiView
    extends BeanPropertyWriter
{
    protected final Class<?>[] _views;

    protected MultiView(BeanPropertyWriter base, Class<?>[] views) {
```

```
            super(base);
            _views = views;
        }

        protected MultiView(MultiView fromView, JsonSerializer<Object> ser) {
            super(fromView, ser);
            _views = fromView._views;
        }

        @Override
        public BeanPropertyWriter withSerializer(JsonSerializer<Object> ser)
        {
            return new MultiView(this, ser);
        }

        @Override
        public void serializeAsField(Object bean, JsonGenerator jgen, SerializerProvider prov)
            throws Exception
        {
            final Class<?> activeView = prov.getSerializationView();
            if (activeView != null) {
                int i = 0, len = _views.length;
                for (; i < len; ++i) {
                    if (_views[i].isAssignableFrom(activeView)) break;
                }
                // not included, bail out:
                if (i == len) {
                    return;
                }
            }
            super.serializeAsField(bean, jgen, prov);
        }
    }
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
import java.util.*;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.ser.impl.PropertySerializerMap;
import org.codehaus.jackson.map.type.TypeFactory;
```

```
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.schema.JsonSchema;
import org.codehaus.jackson.schema.SchemaAware;
import org.codehaus.jackson.type.JavaType;

/**
* Dummy container class to group standard container serializers: serializers
* that can serialize things like {@link java.util.List}s,
* {@link java.util.Map}s and such.
*<p>
* TODO: as per [JACKSON-55], should try to add path info for all serializers;
* is still missing those for some container types.
*/
public final class ContainerSerializers
{
  private ContainerSerializers() { }

  /*
  /**********************************************************
  /* Factory methods
  /**********************************************************
   */

  public static ContainerSerializerBase<?> indexedListSerializer(JavaType elemType,
        boolean staticTyping, TypeSerializer vts, BeanProperty property)
  {
    return new IndexedListSerializer(elemType, staticTyping, vts, property);
  }

  public static ContainerSerializerBase<?> collectionSerializer(JavaType elemType,
        boolean staticTyping, TypeSerializer vts, BeanProperty property)
  {
    return new CollectionSerializer(elemType, staticTyping, vts, property);
  }

  public static ContainerSerializerBase<?> iteratorSerializer(JavaType elemType,
        boolean staticTyping, TypeSerializer vts, BeanProperty property)
  {
    return new IteratorSerializer(elemType, staticTyping, vts, property);
  }

  public static ContainerSerializerBase<?> iterableSerializer(JavaType elemType,
        boolean staticTyping, TypeSerializer vts, BeanProperty property)
  {
    return new IterableSerializer(elemType, staticTyping, vts, property);
  }

  public static JsonSerializer<?> enumSetSerializer(JavaType enumType, BeanProperty property)
```

```
{
    return new EnumSetSerializer(enumType, property);
}

/*
/**********************************************************
/* Base classes
/**********************************************************
 */

/**
 * Base class for serializers that will output contents as JSON
 * arrays.
 */
public abstract static class AsArraySerializer<T>
    extends ContainerSerializerBase<T>
    implements ResolvableSerializer
{
    protected final boolean _staticTyping;

    protected final JavaType _elementType;

    /**
     * Type serializer used for values, if any.
     */
    protected final TypeSerializer _valueTypeSerializer;

    /**
     * Value serializer to use, if it can be statically determined
     *
     * @since 1.5
     */
    protected JsonSerializer<Object> _elementSerializer;

    /**
     * Collection-valued property being serialized with this instance
     *
     * @since 1.7
     */
    protected final BeanProperty _property;

    /**
     * If element type can not be statically determined, mapping from
     * runtime type to serializer is handled using this object
     *
     * @since 1.7
     */
    protected PropertySerializerMap _dynamicSerializers;
```

```
protected AsArraySerializer(Class<?> cls, JavaType et, boolean staticTyping,
        TypeSerializer vts, BeanProperty property)
{
    // typing with generics is messy... have to resort to this:
    super(cls, false);
    _elementType = et;
    // static if explicitly requested, or if element type is final
    _staticTyping = staticTyping || (et != null && et.isFinal());
    _valueTypeSerializer = vts;
    _property = property;
    _dynamicSerializers = PropertySerializerMap.emptyMap();
}

@Override
public final void serialize(T value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
{
    jgen.writeStartArray();
    serializeContents(value, jgen, provider);
    jgen.writeEndArray();
}

@Override
public final void serializeWithType(T value, JsonGenerator jgen, SerializerProvider provider,
        TypeSerializer typeSer)
    throws IOException, JsonGenerationException
{
    typeSer.writeTypePrefixForArray(value, jgen);
    serializeContents(value, jgen, provider);
    typeSer.writeTypeSuffixForArray(value, jgen);
}

protected abstract void serializeContents(T value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException;

@Override
public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    throws JsonMappingException
{
    /* 15-Jan-2010, tatu: This should probably be rewritten, given that
     *    more information about content type is actually being explicitly
     *    passed. So there should be less need to try to re-process that
     *    information.
     */
    ObjectNode o = createSchemaNode("array", true);
    JavaType contentType = null;
    if (typeHint != null) {
```

```
            JavaType javaType = TypeFactory.type(typeHint);
            contentType = javaType.getContentType();
            if (contentType == null) { // could still be parametrized (Iterators)
                if (typeHint instanceof ParameterizedType) {
                    Type[] typeArgs = ((ParameterizedType) typeHint).getActualTypeArguments();
                    if (typeArgs.length == 1) {
                        contentType = TypeFactory.type(typeArgs[0]);
                    }
                }
            }
        }
        if (contentType == null && _elementType != null) {
            contentType = _elementType;
        }
        if (contentType != null) {
            JsonNode schemaNode = null;
            // 15-Oct-2010, tatu: We can't serialize plain Object.class; but what should it produce here? Untyped?
            if (contentType.getRawClass() != Object.class) {
                JsonSerializer<Object> ser = provider.findValueSerializer(contentType, _property);
                if (ser instanceof SchemaAware) {
                    schemaNode = ((SchemaAware) ser).getSchema(provider, null);
                }
            }
            if (schemaNode == null) {
                schemaNode = JsonSchema.getDefaultSchemaNode();
            }
            o.put("items", schemaNode);
        }
        return o;
    }

    /**
     * Need to get callback to resolve value serializer, if static typing
     * is used (either being forced, or because value type is final)
     */
    //@Override
    public void resolve(SerializerProvider provider)
        throws JsonMappingException
    {
        if (_staticTyping && _elementType != null) {
            _elementSerializer = provider.findValueSerializer(_elementType, _property);
        }
    }

    /**
     * @since 1.7
     */
    protected final JsonSerializer<Object> _findAndAddDynamic(PropertySerializerMap map,
```

```
            Class<?> type, SerializerProvider provider) throws JsonMappingException
        {
            PropertySerializerMap.SerializerAndMapResult result = map.findAndAddSerializer(type, provider,
_property);
            // did we get a new map of serializers? If so, start using it
            if (map != result.map) {
                _dynamicSerializers = result.map;
            }
            return result.serializer;
        }

        /**
         * @since 1.8
         */
        protected final JsonSerializer<Object> _findAndAddDynamic(PropertySerializerMap map,
            JavaType type, SerializerProvider provider) throws JsonMappingException
        {
            PropertySerializerMap.SerializerAndMapResult result = map.findAndAddSerializer(type, provider,
_property);
            if (map != result.map) {
                _dynamicSerializers = result.map;
            }
            return result.serializer;
        }
    }

    /*
    /**********************************************************
    /* Concrete serializers, Lists/collections
    /**********************************************************
     */

    /**
     * This is an optimized serializer for Lists that can be efficiently
     * traversed by index (as opposed to others, such as {@link LinkedList}
     * that can not}.
     */
    @JacksonStdImpl
    public static class IndexedListSerializer
        extends AsArraySerializer<List<?>>
    {
        public IndexedListSerializer(JavaType elemType, boolean staticTyping, TypeSerializer vts,
            BeanProperty property)
        {
            super(List.class, elemType, staticTyping, vts, property);
        }

        @Override
```

```
public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts) {
   return new IndexedListSerializer(_elementType, _staticTyping, vts, _property);
}

@Override
public void serializeContents(List<?> value, JsonGenerator jgen, SerializerProvider provider)
   throws IOException, JsonGenerationException
{
   if (_elementSerializer != null) {
      serializeContentsUsing(value, jgen, provider, _elementSerializer);
      return;
   }
   if (_valueTypeSerializer != null) {
      serializeTypedContents(value, jgen, provider);
      return;
   }
   final int len = value.size();
   if (len == 0) {
      return;
   }
   int i = 0;
   try {
      PropertySerializerMap serializers = _dynamicSerializers;
      for (; i < len; ++i) {
         Object elem = value.get(i);
         if (elem == null) {
            provider.defaultSerializeNull(jgen);
         } else {
            Class<?> cc = elem.getClass();
            JsonSerializer<Object> serializer = serializers.serializerFor(cc);
            if (serializer == null) {
               // To fix [JACKSON-508]
               if (_elementType.hasGenericTypes()) {
                  serializer = _findAndAddDynamic(serializers, _elementType.forcedNarrowBy(cc), provider);
               } else {
                  serializer = _findAndAddDynamic(serializers, cc, provider);
               }
            }
            serializer.serialize(elem, jgen, provider);
         }
      }
   } catch (Exception e) {
      // [JACKSON-55] Need to add reference information
      wrapAndThrow(provider, e, value, i);
   }
}

public void serializeContentsUsing(List<?> value, JsonGenerator jgen, SerializerProvider provider,
```

```
        JsonSerializer<Object> ser)
    throws IOException, JsonGenerationException
{
    final int len = value.size();
    if (len == 0) {
        return;
    }
    final TypeSerializer typeSer = _valueTypeSerializer;
    for (int i = 0; i < len; ++i) {
        Object elem = value.get(i);
        try {
            if (elem == null) {
                provider.defaultSerializeNull(jgen);
            } else if (typeSer == null) {
                ser.serialize(elem, jgen, provider);
            } else {
                ser.serializeWithType(elem, jgen, provider, typeSer);
            }
        } catch (Exception e) {
            // [JACKSON-55] Need to add reference information
            wrapAndThrow(provider, e, value, i);
        }
    }
}

public void serializeTypedContents(List<?> value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
{
    final int len = value.size();
    if (len == 0) {
        return;
    }
    int i = 0;
    try {
        final TypeSerializer typeSer = _valueTypeSerializer;
        PropertySerializerMap serializers = _dynamicSerializers;
        for (; i < len; ++i) {
            Object elem = value.get(i);
            if (elem == null) {
                provider.defaultSerializeNull(jgen);
            } else {
                Class<?> cc = elem.getClass();
                JsonSerializer<Object> serializer = serializers.serializerFor(cc);
                if (serializer == null) {
                    // To fix [JACKSON-508]
                    if (_elementType.hasGenericTypes()) {
                        serializer = _findAndAddDynamic(serializers, _elementType.forcedNarrowBy(cc), provider);
                    } else {
```

```
                    serializer = _findAndAddDynamic(serializers, cc, provider);
                }
            }
            serializer.serializeWithType(elem, jgen, provider, typeSer);
          }
        }
      } catch (Exception e) {
        // [JACKSON-55] Need to add reference information
        wrapAndThrow(provider, e, value, i);
      }
    }
}

/**
 * Fallback serializer for cases where Collection is not known to be
 * of type for which more specializer serializer exists (such as
 * index-accessible List).
 * If so, we will just construct an {@link java.util.Iterator}
 * to iterate over elements.
 */
@JacksonStdImpl
public static class CollectionSerializer
    extends AsArraySerializer<Collection<?>>
{
    public CollectionSerializer(JavaType elemType, boolean staticTyping, TypeSerializer vts,
            BeanProperty property)
    {
        super(Collection.class, elemType, staticTyping, vts, property);
    }

    @Override
    public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts) {
        return new CollectionSerializer(_elementType, _staticTyping, vts, _property);
    }

    @Override
    public void serializeContents(Collection<?> value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        if (_elementSerializer != null) {
            serializeContentsUsing(value, jgen, provider, _elementSerializer);
            return;
        }
        Iterator<?> it = value.iterator();
        if (!it.hasNext()) {
            return;
        }
        PropertySerializerMap serializers = _dynamicSerializers;
```

```
    final TypeSerializer typeSer = _valueTypeSerializer;

    int i = 0;
    try {
        do {
            Object elem = it.next();
            if (elem == null) {
                provider.defaultSerializeNull(jgen);
            } else {
                Class<?> cc = elem.getClass();
                JsonSerializer<Object> serializer = serializers.serializerFor(cc);
                if (serializer == null) {
                    // To fix [JACKSON-508]
                    if (_elementType.hasGenericTypes()) {
                        serializer = _findAndAddDynamic(serializers, _elementType.forcedNarrowBy(cc), provider);
                    } else {
                        serializer = _findAndAddDynamic(serializers, cc, provider);
                    }
                }
                if (typeSer == null) {
                    serializer.serialize(elem, jgen, provider);
                } else {
                    serializer.serializeWithType(elem, jgen, provider, typeSer);
                }
            }
            ++i;
        } while (it.hasNext());
    } catch (Exception e) {
        // [JACKSON-55] Need to add reference information
        wrapAndThrow(provider, e, value, i);
    }
}

public void serializeContentsUsing(Collection<?> value, JsonGenerator jgen, SerializerProvider provider,
        JsonSerializer<Object> ser)
    throws IOException, JsonGenerationException
{
    Iterator<?> it = value.iterator();
    if (it.hasNext()) {
        TypeSerializer typeSer = _valueTypeSerializer;
        int i = 0;
        do {
            Object elem = it.next();
            try {
                if (elem == null) {
                    provider.defaultSerializeNull(jgen);
                } else {
                    if (typeSer == null) {
```

```
                    ser.serialize(elem, jgen, provider);
                } else {
                    ser.serializeWithType(elem, jgen, provider, typeSer);
                }
            }
            ++i;
        } catch (Exception e) {
            // [JACKSON-55] Need to add reference information
            wrapAndThrow(provider, e, value, i);
        }
    } while (it.hasNext());
  }
 }
}

@JacksonStdImpl
public static class IteratorSerializer
  extends AsArraySerializer<Iterator<?>>
{
  public IteratorSerializer(JavaType elemType, boolean staticTyping, TypeSerializer vts,
      BeanProperty property)
  {
    super(Iterator.class, elemType, staticTyping, vts, property);
  }

  @Override
  public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts) {
    return new IteratorSerializer(_elementType, _staticTyping, vts, _property);
  }

  @Override
  public void serializeContents(Iterator<?> value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
  {
    if (value.hasNext()) {
      final TypeSerializer typeSer = _valueTypeSerializer;
      JsonSerializer<Object> prevSerializer = null;
      Class<?> prevClass = null;
      do {
        Object elem = value.next();
        if (elem == null) {
          provider.defaultSerializeNull(jgen);
        } else {
          // Minor optimization to avoid most lookups:
          Class<?> cc = elem.getClass();
          JsonSerializer<Object> currSerializer;
          if (cc == prevClass) {
            currSerializer = prevSerializer;
```

```
                } else {
                    currSerializer = provider.findValueSerializer(cc, _property);
                    prevSerializer = currSerializer;
                    prevClass = cc;
                }
                if (typeSer == null) {
                    currSerializer.serialize(elem, jgen, provider);
                } else {
                    currSerializer.serializeWithType(elem, jgen, provider, typeSer);
                }
            }
        } while (value.hasNext());
    }
  }
}

@JacksonStdImpl
public static class IterableSerializer
  extends AsArraySerializer<Iterable<?>>
{
  public IterableSerializer(JavaType elemType, boolean staticTyping, TypeSerializer vts, BeanProperty property)
  {
    super(Iterable.class, elemType, staticTyping, vts, property);
  }

  @Override
  public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts) {
    return new IterableSerializer(_elementType, _staticTyping, vts, _property);
  }

  @Override
  public void serializeContents(Iterable<?> value, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
  {
    Iterator<?> it = value.iterator();
    if (it.hasNext()) {
      final TypeSerializer typeSer = _valueTypeSerializer;
      JsonSerializer<Object> prevSerializer = null;
      Class<?> prevClass = null;

      do {
        Object elem = it.next();
        if (elem == null) {
          provider.defaultSerializeNull(jgen);
        } else {
          // Minor optimization to avoid most lookups:
          Class<?> cc = elem.getClass();
          JsonSerializer<Object> currSerializer;
```

```
                if (cc == prevClass) {
                    currSerializer = prevSerializer;
                } else {
                    currSerializer = provider.findValueSerializer(cc, _property);
                    prevSerializer = currSerializer;
                    prevClass = cc;
                }
                if (typeSer == null) {
                    currSerializer.serialize(elem, jgen, provider);
                } else {
                    currSerializer.serializeWithType(elem, jgen, provider, typeSer);
                }
            }
        } while (it.hasNext());
    }
  }
}

  public static class EnumSetSerializer
    extends AsArraySerializer<EnumSet<? extends Enum<?>>>
  {
    public EnumSetSerializer(JavaType elemType, BeanProperty property)
    {
      super(EnumSet.class, elemType, true, null, property);
    }

    @Override
    public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts) {
      // no typing for enums (always "hard" type)
      return this;
    }

    @Override
    public void serializeContents(EnumSet<? extends Enum<?>> value, JsonGenerator jgen, SerializerProvider
provider)
      throws IOException, JsonGenerationException
    {
      JsonSerializer<Object> enumSer = _elementSerializer;
      /* Need to dynamically find instance serializer; unfortunately
       * that seems to be the only way to figure out type (no accessors
       * to the enum class that set knows)
       */
      for (Enum<?> en : value) {
        if (enumSer == null) {
          /* 12-Jan-2010, tatu: Since enums can not be polymorphic, let's
           *   not bother with typed serializer variant here
           */
          enumSer = provider.findValueSerializer(en.getDeclaringClass(), _property);
```

```
            }
            enumSer.serialize(en, jgen, provider);
          }
        }
      }
    }
    package org.codehaus.jackson.map.ser;

    import java.io.IOException;
    import java.lang.reflect.Type;

    import org.codehaus.jackson.*;
    import org.codehaus.jackson.node.ObjectNode;
    import org.codehaus.jackson.map.*;
    import org.codehaus.jackson.map.annotate.JacksonStdImpl;

    /**
    * Dummy container class to group standard homogenous array serializer implementations
    * (primitive arrays and String array).
    */
    public final class ArraySerializers
    {
      private ArraySerializers() { }

      /*
      /**********************************************************
      /* Base classes
      /**********************************************************
       */

      /**
       * Base class for serializers that will output contents as JSON
       * arrays.
       */
      public abstract static class AsArraySerializer<T>
          extends ContainerSerializerBase<T>
      {
          /**
           * Type serializer used for values, if any.
           */
          protected final TypeSerializer _valueTypeSerializer;

          /**
           * Array-valued property being serialized with this instance
           *
           * @since 1.7
           */
          protected final BeanProperty _property;
```

```
    protected AsArraySerializer(Class<T> cls, TypeSerializer vts, BeanProperty property)
    {
        super(cls);
        _valueTypeSerializer = vts;
        _property = property;
    }

    @Override
    public final void serialize(T value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeStartArray();
        serializeContents(value, jgen, provider);
        jgen.writeEndArray();
    }

    @Override
    public final void serializeWithType(T value, JsonGenerator jgen, SerializerProvider provider,
            TypeSerializer typeSer)
        throws IOException, JsonGenerationException
    {
        typeSer.writeTypePrefixForArray(value, jgen);
        serializeContents(value, jgen, provider);
        typeSer.writeTypeSuffixForArray(value, jgen);
    }

    protected abstract void serializeContents(T value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException;
}

/*
 ***********************************************************
/* Concrete serializers, arrays
 ***********************************************************
 */

/**
 * Standard serializer used for <code>String[]</code> values.
 */
@JacksonStdImpl
public final static class StringArraySerializer
    extends AsArraySerializer<String[]>
    implements ResolvableSerializer
{
    /**
     * Value serializer to use, if it's not the standard one
     * (if it is we can optimize serialization a lot)
```

```
 *
 * @since 1.7
 */
protected JsonSerializer<Object> _elementSerializer;

public StringArraySerializer(BeanProperty prop) {
   super(String[].class, null, prop);
}

/**
 * Strings never add type info; hence, even if type serializer is suggested,
 * we'll ignore it...
 */
@Override
public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts) {
   return this;
}

@Override
public void serializeContents(String[] value, JsonGenerator jgen, SerializerProvider provider)
   throws IOException, JsonGenerationException
{
   final int len = value.length;
   if (len == 0) {
      return;
   }
   if (_elementSerializer != null) {
      serializeContentsSlow(value, jgen, provider, _elementSerializer);
      return;
   }
   /* 08-Dec-2008, tatus: If we want this to be fully overridable
    *  (for example, to support String cleanup during writing
    *  or something), we should find serializer  by provider.
    *  But for now, that seems like an overkill: and caller can
    *  add custom serializer if that is needed as well.
    * (ditto for null values)
    */
   //JsonSerializer<String> ser = (JsonSerializer<String>)provider.findValueSerializer(String.class);
   for (int i = 0; i < len; ++i) {
      String str = value[i];
      if (str == null) {
         jgen.writeNull();
      } else {
         //ser.serialize(value[i], jgen, provider);
         jgen.writeString(value[i]);
      }
   }
}
```

```
    private void serializeContentsSlow(String[] value, JsonGenerator jgen, SerializerProvider provider,
            JsonSerializer<Object> ser)
        throws IOException, JsonGenerationException
    {
        for (int i = 0, len = value.length; i < len; ++i) {
            String str = value[i];
            if (str == null) {
                provider.defaultSerializeNull(jgen);
            } else {
                ser.serialize(value[i], jgen, provider);
            }
        }
    }

    /**
     * Need to get callback to resolve value serializer, which may
     * be overridden by custom serializer
     */
    public void resolve(SerializerProvider provider)
        throws JsonMappingException
    {
        JsonSerializer<Object> ser = provider.findValueSerializer(String.class, _property);
        // Retain if not the standard implementation
        if (ser != null && ser.getClass().getAnnotation(JacksonStdImpl.class) == null) {
            _elementSerializer = ser;
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        ObjectNode o = createSchemaNode("array", true);
        o.put("items", createSchemaNode("string"));
        return o;
    }
}

@JacksonStdImpl
public final static class BooleanArraySerializer
    extends AsArraySerializer<boolean[]>
{
    public BooleanArraySerializer() { super(boolean[].class, null, null); }

    /**
     * Booleans never add type info; hence, even if type serializer is suggested,
     * we'll ignore it...
     */
```

```
    @Override
    public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts)
    {
        return this;
    }

    @Override
    public void serializeContents(boolean[] value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        for (int i = 0, len = value.length; i < len; ++i) {
            jgen.writeBoolean(value[i]);
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        ObjectNode o = createSchemaNode("array", true);
        o.put("items", createSchemaNode("boolean"));
        return o;
    }
}

/**
 * Unlike other integral number array serializers, we do not just print out byte values
 * as numbers. Instead, we assume that it would make more sense to output content
 * as base64 encoded bytes (using default base64 encoding).
 *<p>
 * NOTE: since it is NOT serialized as an array, can not use AsArraySerializer as base
 */
@JacksonStdImpl
public final static class ByteArraySerializer
    extends SerializerBase<byte[]>
{
    public ByteArraySerializer() {
        super(byte[].class);
    }

    @Override
    public void serialize(byte[] value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeBinary(value);
    }

    @Override
    public void serializeWithType(byte[] value, JsonGenerator jgen, SerializerProvider provider,
```

```
            TypeSerializer typeSer)
        throws IOException, JsonGenerationException
    {
        typeSer.writeTypePrefixForScalar(value, jgen);
        jgen.writeBinary(value);
        typeSer.writeTypeSuffixForScalar(value, jgen);
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        ObjectNode o = createSchemaNode("array", true);
        ObjectNode itemSchema = createSchemaNode("string"); //binary values written as strings?
        o.put("items", itemSchema);
        return o;
    }
}

@JacksonStdImpl
public final static class ShortArraySerializer
    extends AsArraySerializer<short[]>
{
    public ShortArraySerializer() { this(null); }
    public ShortArraySerializer(TypeSerializer vts) { super(short[].class, vts, null); }

    @Override
    public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts) {
        return new ShortArraySerializer(vts);
    }

    @SuppressWarnings("cast")
    @Override
    public void serializeContents(short[] value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        for (int i = 0, len = value.length; i < len; ++i) {
            jgen.writeNumber((int)value[i]);
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        //no "short" type defined by json
        ObjectNode o = createSchemaNode("array", true);
        o.put("items", createSchemaNode("integer"));
        return o;
    }
```

```
}

/**
 * Character arrays are different from other integral number arrays in that
 * they are most likely to be textual data, and should be written as
 * Strings, not arrays of entries.
 *<p>
 * NOTE: since it is NOT serialized as an array, can not use AsArraySerializer as base
 */
@JacksonStdImpl
public final static class CharArraySerializer
   extends SerializerBase<char[]>
{
   public CharArraySerializer() { super(char[].class); }

   @Override
   public void serialize(char[] value, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonGenerationException
   {
      // [JACKSON-289] allows serializing as 'sparse' char array too:
      if (provider.isEnabled(SerializationConfig.Feature.WRITE_CHAR_ARRAYS_AS_JSON_ARRAYS)) {
         jgen.writeStartArray();
         _writeArrayContents(jgen, value);
         jgen.writeEndArray();
      } else {
         jgen.writeString(value, 0, value.length);
      }
   }

   @Override
   public void serializeWithType(char[] value, JsonGenerator jgen, SerializerProvider provider,
         TypeSerializer typeSer)
      throws IOException, JsonGenerationException
   {
      // [JACKSON-289] allows serializing as 'sparse' char array too:
      if (provider.isEnabled(SerializationConfig.Feature.WRITE_CHAR_ARRAYS_AS_JSON_ARRAYS)) {
         typeSer.writeTypePrefixForArray(value, jgen);
         _writeArrayContents(jgen, value);
         typeSer.writeTypeSuffixForArray(value, jgen);
      } else { // default is to write as simple String
         typeSer.writeTypePrefixForScalar(value, jgen);
         jgen.writeString(value, 0, value.length);
         typeSer.writeTypeSuffixForScalar(value, jgen);
      }
   }

   private final void _writeArrayContents(JsonGenerator jgen, char[] value)
      throws IOException, JsonGenerationException
```

```
    {
        for (int i = 0, len = value.length; i < len; ++i) {
            jgen.writeString(value, i, 1);
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        ObjectNode o = createSchemaNode("array", true);
        ObjectNode itemSchema = createSchemaNode("string");
        itemSchema.put("type", "string");
        o.put("items", itemSchema);
        return o;
    }
}

@JacksonStdImpl
public final static class IntArraySerializer
    extends AsArraySerializer<int[]>
{
    public IntArraySerializer() { super(int[].class, null, null); }

    /**
     * Ints never add type info; hence, even if type serializer is suggested,
     * we'll ignore it...
     */
    @Override
    public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts)
    {
        return this;
    }

    @Override
    public void serializeContents(int[] value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        for (int i = 0, len = value.length; i < len; ++i) {
            jgen.writeNumber(value[i]);
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        ObjectNode o = createSchemaNode("array", true);
        o.put("items", createSchemaNode("integer"));
        return o;
```

```
    }
}

@JacksonStdImpl
public final static class LongArraySerializer
    extends AsArraySerializer<long[]>
{
    public LongArraySerializer() { this(null); }
    public LongArraySerializer(TypeSerializer vts) { super(long[].class, vts, null); }

    @Override
    public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts) {
        return new LongArraySerializer(vts);
    }

    @Override
    public void serializeContents(long[] value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        for (int i = 0, len = value.length; i < len; ++i) {
            jgen.writeNumber(value[i]);
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        ObjectNode o = createSchemaNode("array", true);
        o.put("items", createSchemaNode("number", true));
        return o;
    }
}

@JacksonStdImpl
public final static class FloatArraySerializer
    extends AsArraySerializer<float[]>
{
    public FloatArraySerializer() { this(null); }
    public FloatArraySerializer(TypeSerializer vts) { super(float[].class, vts, null); }

    @Override
    public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts) {
        return new FloatArraySerializer(vts);
    }

    @Override
    public void serializeContents(float[] value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
```

```
    {
        for (int i = 0, len = value.length; i < len; ++i) {
            jgen.writeNumber(value[i]);
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        ObjectNode o = createSchemaNode("array", true);
        o.put("items", createSchemaNode("number"));
        return o;
    }
}

@JacksonStdImpl
public final static class DoubleArraySerializer
    extends AsArraySerializer<double[]>
{
    public DoubleArraySerializer() { super(double[].class, null, null); }

    /**
     * Doubles never add type info; hence, even if type serializer is suggested,
     * we'll ignore it...
     */
    @Override
    public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts)
    {
        return this;
    }

    @Override
    public void serializeContents(double[] value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        for (int i = 0, len = value.length; i < len; ++i) {
            jgen.writeNumber(value[i]);
        }
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        ObjectNode o = createSchemaNode("array", true);
        o.put("items", createSchemaNode("number"));
        return o;
    }
}
```

```
}
package org.codehaus.jackson.map.ser;

import java.util.List;

import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.SerializationConfig;
import org.codehaus.jackson.map.introspect.BasicBeanDescription;

/**
* Abstract class that defines API for objects that can be registered (for {@link BeanSerializerFactory}
* to participate in constructing {@link BeanSerializer} instances.
* This is typically done by modules that want alter some aspects of serialization
* process; and is preferable to sub-classing of {@link BeanSerializerFactory}.
*<p>
* Sequence in which callback methods are called is as follows:
* <ol>
*  <li>After factory has collected tentative set of properties (instances of
*    <code>BeanPropertyWriter</code>) is sent for modification via
*    {@link #changeProperties}. Changes can include removal, addition and
*    replacement of suggested properties.
*  <li>Resulting set of properties are ordered (sorted) by factory, as per
*    configuration, and then {@link #orderProperties} is called to allow
*    modifiers to alter ordering.
*  <li>After all bean properties and related information is accumulated,
*    {@link #updateBuilder} is called with builder, to allow builder state
*    to be modified (including possibly replacing builder itself if necessary)
*  <li>Once all bean information has been determined,
*    factory creates default {@link BeanSerializer} instance and passes
*    it to modifiers using {@link #modifySerializer}, for possible
*    modification or replacement (by any {@link org.codehaus.jackson.map.JsonSerializer} instance)
* </ol>
*<p>
* Default method implementations are "no-op"s, meaning that methods are implemented
* but have no effect.
*
* @since 1.7
*/
public abstract class BeanSerializerModifier
{
  /**
   * Method called by {@link BeanSerializerFactory} with tentative set
   * of discovered properties.
   * Implementations can add, remove or replace any of passed properties.
   *
   * Properties <code>List</code> passed as argument is modifiable, and returned List must
   * likewise be modifiable as it may be passed to multiple registered
   * modifiers.
```

```
     */
    public List<BeanPropertyWriter> changeProperties(SerializationConfig config,
            BasicBeanDescription beanDesc, List<BeanPropertyWriter> beanProperties) {
        return beanProperties;
    }

    /**
     * Method called by {@link BeanSerializerFactory} with set of properties
     * to serialize, in default ordering (based on defaults as well as
     * possible type annotations).
     * Implementations can change ordering any way they like.
     *
     * Properties <code>List</code> passed as argument is modifiable, and returned List must
     * likewise be modifiable as it may be passed to multiple registered
     * modifiers.
     */
    public List<BeanPropertyWriter> orderProperties(SerializationConfig config,
            BasicBeanDescription beanDesc, List<BeanPropertyWriter> beanProperties) {
        return beanProperties;
    }

    /**
     * Method called by {@link BeanSerializerFactory} after collecting all information
     * regarding POJO to serialize and updating builder with it, but before constructing
     * serializer.
     * Implementations may choose to modify state of builder (to affect serializer being
     * built), or even completely replace it (if they want to build different kind of
     * serializer). Typically, however, passed-in builder is returned, possibly with
     * some modifications.
     */
    public BeanSerializerBuilder updateBuilder(SerializationConfig config,
            BasicBeanDescription beanDesc, BeanSerializerBuilder builder) {
        return builder;
    }

    /**
     * Method called by {@link BeanSerializerFactory} after constructing default
     * bean serializer instance with properties collected and ordered earlier.
     * Implementations can modify or replace given serializer and return serializer
     * to use. Note that although initial serializer being passed is of type
     * {@link BeanSerializer}, modifiers may return serializers of other types;
     * and this is why implementations must check for type before casting.
     */
    public JsonSerializer<?> modifySerializer(SerializationConfig config,
            BasicBeanDescription beanDesc, JsonSerializer<?> serializer) {
        return serializer;
    }
}
```

```
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.Type;
import java.util.Collection;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.schema.SchemaAware;
import org.codehaus.jackson.schema.JsonSchema;
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.type.JavaType;

/**
* Serializer class that can serialize arbitrary bean objects
*<p>
* Implementation note: we will post-process resulting serializer,
* to figure out actual serializers for final types. This must be
* done from {@link #resolve} method, and NOT from constructor;
* otherwise we could end up with an infinite loop.
*<p>
* Since 1.7 instances are immutable; this is achieved by using a
* separate builder during construction process.
*/
public class BeanSerializer
  extends SerializerBase<Object>
  implements ResolvableSerializer, SchemaAware
{
  final protected static BeanPropertyWriter[] NO_PROPS = new BeanPropertyWriter[0];

  /*
  /**********************************************************
  /* Configuration
  /**********************************************************
   */

  /**
   * Writers used for outputting actual property values
   */
  final protected BeanPropertyWriter[] _props;

  /**
   * Optional filters used to suppress output of properties that
   * are only to be included in certain views
   */
```

```
final protected BeanPropertyWriter[] _filteredProps;

/**
 * Handler for {@link org.codehaus.jackson.annotate.JsonAnyGetter}
 * annotated properties
 *
 * @since 1.6
 */
final protected AnyGetterWriter _anyGetterWriter;

/**
 * Id of the bean property filter to use, if any; null if none.
 */
final protected Object _propertyFilterId;

/*
/**********************************************************
/* Life-cycle: constructors
/**********************************************************
 */

/**
 * @param type Nominal type of values handled by this serializer
 * @param properties Property writers used for actual serialization
 */
public BeanSerializer(JavaType type,
        BeanPropertyWriter[] properties, BeanPropertyWriter[] filteredProperties,
        AnyGetterWriter anyGetterWriter,
        Object filterId)
{
    super(type);
    _props = properties;
    _filteredProps = filteredProperties;
    _anyGetterWriter = anyGetterWriter;
    _propertyFilterId = filterId;
}

@SuppressWarnings("unchecked")
public BeanSerializer(Class<?> rawType,
        BeanPropertyWriter[] properties, BeanPropertyWriter[] filteredProperties,
        AnyGetterWriter anyGetterWriter,
        Object filterId)
{
    super((Class<Object>) rawType);
    _props = properties;
    _filteredProps = filteredProperties;
    _anyGetterWriter = anyGetterWriter;
    _propertyFilterId = filterId;
```

```
}

/**
 * Copy-constructor that is useful for sub-classes that just want to
 * copy all super-class properties without modifications.
 *
 * @since 1.7
 */
protected BeanSerializer(BeanSerializer src) {
   this(src._handledType,
         src._props, src._filteredProps, src._anyGetterWriter, src._propertyFilterId);
}

/**
 * @deprecated since 1.7, use method that takes more arguments.
 */
@Deprecated
@SuppressWarnings("unchecked")
public BeanSerializer(Class<?> type, BeanPropertyWriter[] properties, Object filterId)
{
   super((Class<Object>)type);
   _props = properties;
   _filteredProps = null;
   _anyGetterWriter = null;
   _propertyFilterId = filterId;
}

/**
 * @deprecated since 1.7, use method that takes more arguments.
 */
@Deprecated
@SuppressWarnings("unchecked")
public BeanSerializer(Class<?> type, BeanPropertyWriter[] properties,
      BeanPropertyWriter[] filteredProperties)
{
   super((Class<Object>)type);
   _props = properties;
   _filteredProps = filteredProperties;
   _anyGetterWriter = null;
   _propertyFilterId = null;
}

/**
 * @deprecated since 1.7, use method that takes more arguments.
 */
@Deprecated
public BeanSerializer(Class<?> type, Collection<BeanPropertyWriter> props)
{
```

```
    this(type, props.toArray(new BeanPropertyWriter[props.size()]), null, null, null);
  }

  /**
   * @deprecated since 1.7, use method that takes more arguments.
   */
  @Deprecated
  public BeanSerializer(Class<?> type, BeanPropertyWriter[] writers)
  {
    this(type, writers, null, null, null);
  }

  /*
  /**********************************************************
  /* Life-cycle: factory methods, fluent factories
  /**********************************************************
   */

  /**
   * Method used for constructing a filtered serializer instance, using this
   * serializer as the base.
   *
   * @deprecated Since 1.7 {@link BeanSerializerBuilder} should be used,
   *   so that no copy-methods are not needed
   */
  @Deprecated
  public BeanSerializer withFiltered(BeanPropertyWriter[] filtered)
  {
    // 03-Jan-2011, tatu: Need to ensure sub-classes override, so:
    if (getClass() != BeanSerializer.class) {
      throw new IllegalStateException("BeanSerializer.withFiltered() called on base class: sub-classes MUST
override method");
    }
    // if no filters, no need to construct new instance...
    if (filtered == null && _filteredProps == null) {
      return this;
    }
    return new BeanSerializer(handledType(), _props, filtered, _anyGetterWriter, _propertyFilterId);
  }

  /**
   * Method for constructing dummy bean deserializer; one that
   * never outputs any properties
   */
  public static BeanSerializer createDummy(Class<?> forType)
  {
    return new BeanSerializer(forType, NO_PROPS, null, null, null);
  }
```

```
/*
/**********************************************************
/* JsonSerializer implementation
/**********************************************************
 */

/**
 * Main serialization method that will delegate actual output to
 * configured
 * {@link BeanPropertyWriter} instances.
 */
@Override
public final void serialize(Object bean, JsonGenerator jgen, SerializerProvider provider)
   throws IOException, JsonGenerationException
{
   jgen.writeStartObject();
   if (_propertyFilterId != null) {
      serializeFieldsFiltered(bean, jgen, provider);
   } else {
      serializeFields(bean, jgen, provider);
   }
   jgen.writeEndObject();
}

@Override
public void serializeWithType(Object bean, JsonGenerator jgen, SerializerProvider provider,
      TypeSerializer typeSer)
   throws IOException, JsonGenerationException
{
   typeSer.writeTypePrefixForObject(bean, jgen);
   if (_propertyFilterId != null) {
      serializeFieldsFiltered(bean, jgen, provider);
   } else {
      serializeFields(bean, jgen, provider);
   }
   typeSer.writeTypeSuffixForObject(bean, jgen);
}

protected void serializeFields(Object bean, JsonGenerator jgen, SerializerProvider provider)
   throws IOException, JsonGenerationException
{
   final BeanPropertyWriter[] props;
   if (_filteredProps != null && provider.getSerializationView() != null) {
      props = _filteredProps;
   } else {
      props = _props;
   }
```

```
    int i = 0;
    try {
        for (final int len = props.length; i < len; ++i) {
            BeanPropertyWriter prop = props[i];
            if (prop != null) { // can have nulls in filtered list
                prop.serializeAsField(bean, jgen, provider);
            }
        }
        if (_anyGetterWriter != null) {
            _anyGetterWriter.getAndSerialize(bean, jgen, provider);
        }
    } catch (Exception e) {
        String name = (i == props.length) ? "[anySetter]" : props[i].getName();
        wrapAndThrow(provider, e, bean, name);
    } catch (StackOverflowError e) {
        /* 04-Sep-2009, tatu: Dealing with this is tricky, since we do not
         *   have many stack frames to spare... just one or two; can't
         *   make many calls.
         */
        JsonMappingException mapE = new JsonMappingException("Infinite recursion (StackOverflowError)");
        String name = (i == props.length) ? "[anySetter]" : props[i].getName();
        mapE.prependPath(new JsonMappingException.Reference(bean, name));
        throw mapE;
    }
}

/**
 * Alternative serialization method that gets called when there is a
 * {@link BeanPropertyFilter} that needs to be called to determine
 * which properties are to be serialized (and possibly how)
 *
 * @since 1.7
 */
protected void serializeFieldsFiltered(Object bean, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
{
    /* note: almost verbatim copy of "serializeFields"; copied (instead of merged)
     * so that old method need not add check for existence of filter.
     */

    final BeanPropertyWriter[] props;
    if (_filteredProps != null && provider.getSerializationView() != null) {
        props = _filteredProps;
    } else {
        props = _props;
    }
    final BeanPropertyFilter filter = findFilter(provider);
    int i = 0;
```

```
        try {
            for (final int len = props.length; i < len; ++i) {
                BeanPropertyWriter prop = props[i];
                if (prop != null) { // can have nulls in filtered list
                    filter.serializeAsField(bean, jgen, provider, prop);
                }
            }
            if (_anyGetterWriter != null) {
                _anyGetterWriter.getAndSerialize(bean, jgen, provider);
            }
        } catch (Exception e) {
            String name = (i == props.length) ? "[anySetter]" : props[i].getName();
            wrapAndThrow(provider, e, bean, name);
        } catch (StackOverflowError e) {
            JsonMappingException mapE = new JsonMappingException("Infinite recursion (StackOverflowError)");
            String name = (i == props.length) ? "[anySetter]" : props[i].getName();
            mapE.prependPath(new JsonMappingException.Reference(bean, name));
            throw mapE;
        }
    }

    /**
     * Helper method used to locate filter that is needed, based on filter id
     * this serializer was constructed with.
     *
     * @since 1.7
     */
    protected BeanPropertyFilter findFilter(SerializerProvider provider)
        throws JsonMappingException
    {
        final Object filterId = _propertyFilterId;
        FilterProvider filters = provider.getFilterProvider();
        // Not ok to miss the provider, if a filter is declared to be needed!
        if (filters == null) {
            throw new JsonMappingException("Can not resolve BeanPropertyFilter with id '"+filterId+"'; no
FilterProvider configured");
        }
        BeanPropertyFilter filter = filters.findFilter(filterId);
        // But is it ok not to find a filter? For now let's assume it is not; can add a feature to disable errors if need be
        if (filter == null) {
            throw new JsonMappingException("No filter configured with id '"+filterId+"' (type "
                    +filterId.getClass().getName()+")");
        }
        return filter;
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
```

```
    throws JsonMappingException
{
    ObjectNode o = createSchemaNode("object", true);
    //todo: should the classname go in the title?
    //o.put("title", _className);
    ObjectNode propertiesNode = o.objectNode();
    for (int i = 0; i < _props.length; i++) {
        BeanPropertyWriter prop = _props[i];
        JavaType propType = prop.getSerializationType();
        // 03-Dec-2010, tatu: SchemaAware REALLY should use JavaType, but alas it doesn't...
        Type hint = (propType == null) ? prop.getGenericPropertyType() : propType.getRawClass();
        // Maybe it already has annotated/statically configured serializer?
        JsonSerializer<Object> ser = prop.getSerializer();
        if (ser == null) { // nope
            Class<?> serType = prop.getRawSerializationType();
            if (serType == null) {
                serType = prop.getPropertyType();
            }
            ser = provider.findValueSerializer(serType, prop);
        }
        JsonNode schemaNode = (ser instanceof SchemaAware) ?
                ((SchemaAware) ser).getSchema(provider, hint) :
                JsonSchema.getDefaultSchemaNode();
        propertiesNode.put(prop.getName(), schemaNode);
    }
    o.put("properties", propertiesNode);
    return o;
}

/*
/**********************************************************
/* ResolvableSerializer impl
/**********************************************************
 */

public void resolve(SerializerProvider provider)
    throws JsonMappingException
{
    //AnnotationIntrospector ai = provider.getConfig().getAnnotationIntrospector();
    int filteredCount = (_filteredProps == null) ? 0 : _filteredProps.length;
    for (int i = 0, len = _props.length; i < len; ++i) {
        BeanPropertyWriter prop = _props[i];
        if (prop.hasSerializer()) {
            continue;
        }
        // Was the serialization type hard-coded? If so, use it
        JavaType type = prop.getSerializationType();
```

```
            /* It not, we can use declared return type if and only if
             * declared type is final -- if not, we don't really know
             * the actual type until we get the instance.
             */
            if (type == null) {
                type = TypeFactory.type(prop.getGenericPropertyType());
                if (!type.isFinal()) {
                    /* 18-Feb-2010, tatus: But even if it is non-final,
                     *    we may need to retain some of type information
                     *    so that we can accurately handle contained
                     *    types
                     */
                    if (type.isContainerType() || type.containedTypeCount() > 0) {
                        prop.setNonTrivialBaseType(type);
                    }
                    continue;
                }
            }
            JsonSerializer<Object> ser = provider.findValueSerializer(type, prop);
            /* 04-Feb-2010, tatu: We may have stashed type serializer for content types
             *   too, earlier; if so, it's time to connect the dots here:
             */
            if (type.isContainerType()) {
                TypeSerializer typeSer = type.getContentType().getTypeHandler();
                if (typeSer != null) {
                    // for now, can do this only for standard containers...
                    if (ser instanceof ContainerSerializerBase<?>) {
                        // ugly casts... but necessary
                        @SuppressWarnings("unchecked")
                        JsonSerializer<Object> ser2 = (JsonSerializer<Object>)((ContainerSerializerBase<?>)
ser).withValueTypeSerializer(typeSer);
                        ser = ser2;
                    }
                }
            }
            prop = prop.withSerializer(ser);
            _props[i] = prop;
            // and maybe replace filtered property too? (see [JACKSON-364])
            if (i < filteredCount) {
                BeanPropertyWriter w2 = _filteredProps[i];
                if (w2 != null) {
                    _filteredProps[i] = w2.withSerializer(ser);
                }
            }
        }

        // also, any-getter may need to be resolved
        if (_anyGetterWriter != null) {
```

```
            _anyGetterWriter.resolve(provider);
        }
    }

    /*
    /**********************************************************
    /* Standard methods
    /**********************************************************
     */

    @Override public String toString() {
        return "BeanSerializer for "+handledType().getName();
    }
}
package org.codehaus.jackson.map.ser;

import java.lang.reflect.Field;
import java.lang.reflect.Method;

import org.codehaus.jackson.map.AnnotationIntrospector;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.SerializationConfig;
import org.codehaus.jackson.map.TypeSerializer;
import org.codehaus.jackson.map.annotate.JsonSerialize;
import org.codehaus.jackson.map.annotate.JsonSerialize.Inclusion;
import org.codehaus.jackson.map.introspect.Annotated;
import org.codehaus.jackson.map.introspect.AnnotatedField;
import org.codehaus.jackson.map.introspect.AnnotatedMember;
import org.codehaus.jackson.map.introspect.AnnotatedMethod;
import org.codehaus.jackson.map.introspect.BasicBeanDescription;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.map.util.Annotations;
import org.codehaus.jackson.type.JavaType;

/**
* Helper class for {@link BeanSerializerFactory} that is used to
* construct {@link BeanPropertyWriter} instances. Can be sub-classed
* to change behavior.
*/
public class PropertyBuilder
{
    final protected SerializationConfig _config;
    final protected BasicBeanDescription _beanDesc;
    final protected JsonSerialize.Inclusion _outputProps;

    final protected AnnotationIntrospector _annotationIntrospector;

    /**
```

```
     * If a property has serialization inclusion value of
     * {@link Inclusion#ALWAYS}, we need to know the default
     * value of the bean, to know if property value equals default
     * one.
     */
    protected Object _defaultBean;

    public PropertyBuilder(SerializationConfig config, BasicBeanDescription beanDesc)
    {
        _config = config;
        _beanDesc = beanDesc;
        _outputProps = beanDesc.findSerializationInclusion(config.getSerializationInclusion());
        _annotationIntrospector = _config.getAnnotationIntrospector();
    }

    /*
    /**********************************************************
    /* Public API
    /**********************************************************
     */

    public Annotations getClassAnnotations() {
        return _beanDesc.getClassAnnotations();
    }

    /**
     * @param contentTypeSer Optional explicit type information serializer
     *    to use for contained values (only used for properties that are
     *    of container type)
     */
    protected BeanPropertyWriter buildWriter(String name, JavaType declaredType,
            JsonSerializer<Object> ser,
            TypeSerializer typeSer, TypeSerializer contentTypeSer,
            AnnotatedMember am, boolean defaultUseStaticTyping)
    {
        Field f;
        Method m;
        if (am instanceof AnnotatedField) {
            m = null;
            f = ((AnnotatedField) am).getAnnotated();
        } else {
            m = ((AnnotatedMethod) am).getAnnotated();
            f = null;
        }

        // do we have annotation that forces type to use (to declared type or its super type)?
        JavaType serializationType = findSerializationType(am, defaultUseStaticTyping);
```

```
        // Container types can have separate type serializers for content (value / element) type
        if (contentTypeSer != null) {
            /* 04-Feb-2010, tatu: Let's force static typing for collection, if there is
             *    type information for contents. Should work well (for JAXB case); can be
             *    revisited if this causes problems.
             */
            if (serializationType == null) {
//                serializationType = TypeFactory.type(am.getGenericType(), _beanDesc.getType());
                serializationType = declaredType;
            }
            JavaType ct = serializationType.getContentType();
            /* 03-Sep-2010, tatu: This is somehow related to [JACKSON-356], but I don't completely
             *   yet understand how pieces fit together. Still, better be explicit than rely on
             *   NPE to indicate an issue...
             */
            if (ct == null) {
                throw new IllegalStateException("Problem trying to create BeanPropertyWriter for property '"
                        +name+"' (of type "+_beanDesc.getType()+"); serialization type "+serializationType+" has no
content");
            }
            serializationType = serializationType.withContentTypeHandler(contentTypeSer);
            ct = serializationType.getContentType();
        }
        Object suppValue = null;
        boolean suppressNulls = false;

        JsonSerialize.Inclusion methodProps = _annotationIntrospector.findSerializationInclusion(am, _outputProps);

        if (methodProps != null) {
            switch (methodProps) {
            case NON_DEFAULT:
                suppValue = getDefaultValue(name, m, f);
                if (suppValue == null) {
                    suppressNulls = true;
                }
                break;
            case NON_NULL:
                suppressNulls = true;
                break;
            }
        }
        return new BeanPropertyWriter(am, _beanDesc.getClassAnnotations(), name, declaredType,
                ser, typeSer, serializationType, m, f, suppressNulls, suppValue);
    }

    /*
    /**********************************************************
    /* Helper methods, generic
```

```
    /**********************************************************
     */

    /**
     * Method that will try to determine statically defined type of property
     * being serialized, based on annotations (for overrides), and alternatively
     * declared type (if static typing for serialization is enabled).
     * If neither can be used (no annotations, dynamic typing), returns null.
     */
    protected JavaType findSerializationType(Annotated a, boolean useStaticTyping)
    {
        // [JACKSON-120]: Check to see if serialization type is fixed
        Class<?> serializationType = _annotationIntrospector.findSerializationType(a);
        if (serializationType != null) {
            // Must be a super type to be usable
            Class<?> raw = a.getRawType();
            if (serializationType.isAssignableFrom(raw)) {
                return TypeFactory.type(serializationType);
            }
            /* 18-Nov-2010, tatu: Related to fixing [JACKSON-416], an issue with such
             *   check is that for deserialization more specific type makes sense;
             *   and for serialization more generic. But alas JAXB uses but a single
             *   annotation to do both... Hence, we must just discard type, as long as
             *   types are related
             */
            if (!raw.isAssignableFrom(serializationType)) {
                throw new IllegalArgumentException("Illegal concrete-type annotation for method '"+a.getName()+"':
class "+serializationType.getName()+" not a super-type of (declared) class "+raw.getName());
            }
            /* 03-Dec-2010, tatu: Actually, ugh, to resolve [JACKSON-415] may further relax this
             *   and actually accept subtypes too for serialization. Bit dangerous in theory
             *   but need to trust user here...
             */
            return TypeFactory.type(serializationType);
        }
        /* [JACKSON-114]: if using static typing, declared type is known
         * to be the type...
         */
        JsonSerialize.Typing typing = _annotationIntrospector.findSerializationTyping(a);
        if (typing != null) {
            useStaticTyping = (typing == JsonSerialize.Typing.STATIC);
        }
        if (useStaticTyping) {
            return TypeFactory.type(a.getGenericType(), _beanDesc.getType());
        }
        return null;
    }
```

```
  protected Object getDefaultBean()
  {
     if (_defaultBean == null) {
        /* If we can fix access rights, we should; otherwise non-public
         * classes or default constructor will prevent instantiation
         */
        _defaultBean =
_beanDesc.instantiateBean(_config.isEnabled(SerializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIE
RS));
        if (_defaultBean == null) {
           Class<?> cls = _beanDesc.getClassInfo().getAnnotated();
           throw new IllegalArgumentException("Class "+cls.getName()+" has no default constructor; can not
instantiate default bean value to support 'properties=JsonSerialize.Inclusion.NON_DEFAULT' annotation");
        }
     }
     return _defaultBean;
  }

  protected Object getDefaultValue(String name, Method m, Field f)
  {
     Object defaultBean = getDefaultBean();
     try {
        if (m != null) {
           return m.invoke(defaultBean);
        }
        return f.get(defaultBean);
     } catch (Exception e) {
        return _throwWrapped(e, name, defaultBean);
     }
  }

  protected Object _throwWrapped(Exception e, String propName, Object defaultBean)
  {
     Throwable t = e;
     while (t.getCause() != null) {
        t = t.getCause();
     }
     if (t instanceof Error) throw (Error) t;
     if (t instanceof RuntimeException) throw (RuntimeException) t;
     throw new IllegalArgumentException("Failed to get property '"+propName+"' of default
"+defaultBean.getClass().getName()+" instance");
  }
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
```

```
import java.util.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.map.util.EnumValues;
import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.node.JsonNodeFactory;
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.schema.JsonSchema;
import org.codehaus.jackson.schema.SchemaAware;

/**
* Specialized serializer for {@link EnumMap}s. Somewhat tricky to
* implement because actual Enum value type may not be available;
* and if not, it can only be gotten from actual instance.
*/
@JacksonStdImpl
public class EnumMapSerializer
  extends ContainerSerializerBase<EnumMap<? extends Enum<?>, ?>>
  implements ResolvableSerializer
{
  protected final boolean _staticTyping;

  /**
   * If we know enumeration used as key, this will contain
   * value set to use for serialization
   */
  protected final EnumValues _keyEnums;

  protected final JavaType _valueType;

  /**
   * Property being serialized with this instance
   *
   * @since 1.7
   */
  protected final BeanProperty _property;

  /**
   * Value serializer to use, if it can be statically determined
   *
   * @since 1.5
   */
  protected JsonSerializer<Object> _valueSerializer;

  /**
```

```
     * Type serializer used for values, if any.
     */
    protected final TypeSerializer _valueTypeSerializer;

    public EnumMapSerializer(JavaType valueType, boolean staticTyping, EnumValues keyEnums,
            TypeSerializer vts, BeanProperty property)
    {
        super(EnumMap.class, false);
        _staticTyping = staticTyping || (valueType != null && valueType.isFinal());
        _valueType = valueType;
        _keyEnums = keyEnums;
        _valueTypeSerializer = vts;
        _property = property;
    }

    @Override
    public ContainerSerializerBase<?> _withValueTypeSerializer(TypeSerializer vts)
    {
        return new EnumMapSerializer(_valueType, _staticTyping, _keyEnums, vts,  _property);
    }

    @Override
    public void serialize(EnumMap<? extends Enum<?>,?> value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeStartObject();
        if (!value.isEmpty()) {
            serializeContents(value, jgen, provider);
        }
        jgen.writeEndObject();
    }

    @Override
    public void serializeWithType(EnumMap<? extends Enum<?>,?> value, JsonGenerator jgen, SerializerProvider
provider,
            TypeSerializer typeSer)
        throws IOException, JsonGenerationException
    {
        typeSer.writeTypePrefixForObject(value, jgen);
        if (!value.isEmpty()) {
            serializeContents(value, jgen, provider);
        }
        typeSer.writeTypeSuffixForObject(value, jgen);
    }

    protected void serializeContents(EnumMap<? extends Enum<?>,?> value, JsonGenerator jgen, SerializerProvider
provider)
        throws IOException, JsonGenerationException
```

```
{
    if (_valueSerializer != null) {
        serializeContentsUsing(value, jgen, provider, _valueSerializer);
        return;
    }
    JsonSerializer<Object> prevSerializer = null;
    Class<?> prevClass = null;
    EnumValues keyEnums = _keyEnums;

    for (Map.Entry<? extends Enum<?>,?> entry : value.entrySet()) {
        // First, serialize key
        Enum<?> key = entry.getKey();
        if (keyEnums == null) {
            /* 15-Oct-2009, tatu: This is clumsy, but still the simplest efficient
             * way to do it currently, as Serializers get cached. (it does assume we'll always use
             * default serializer tho -- so ideally code should be rewritten)
             */
            // ... and lovely two-step casting process too...
            SerializerBase<?> ser = (SerializerBase<?>) provider.findValueSerializer(
                    key.getDeclaringClass(), _property);
            keyEnums = ((EnumSerializer) ser).getEnumValues();
        }
        jgen.writeFieldName(keyEnums.serializedValueFor(key));
        // And then value
        Object valueElem = entry.getValue();
        if (valueElem == null) {
            provider.defaultSerializeNull(jgen);
        } else {
            Class<?> cc = valueElem.getClass();
            JsonSerializer<Object> currSerializer;
            if (cc == prevClass) {
                currSerializer = prevSerializer;
            } else {
                currSerializer = provider.findValueSerializer(cc, _property);
                prevSerializer = currSerializer;
                prevClass = cc;
            }
            try {
                currSerializer.serialize(valueElem, jgen, provider);
            } catch (Exception e) {
                // [JACKSON-55] Need to add reference information
                wrapAndThrow(provider, e, value, entry.getKey().name());
            }
        }
    }
}

protected void serializeContentsUsing(EnumMap<? extends Enum<?>,?> value, JsonGenerator jgen,
```

```
SerializerProvider provider,
        JsonSerializer<Object> valueSer)
    throws IOException, JsonGenerationException
  {
    EnumValues keyEnums = _keyEnums;
    for (Map.Entry<? extends Enum<?>,?> entry : value.entrySet()) {
      Enum<?> key = entry.getKey();
      if (keyEnums == null) {
        // clumsy, but has to do for now:
        SerializerBase<?> ser = (SerializerBase<?>) provider.findValueSerializer(key.getDeclaringClass(),
            _property);
        keyEnums = ((EnumSerializer) ser).getEnumValues();
      }
      jgen.writeFieldName(keyEnums.serializedValueFor(key));
      Object valueElem = entry.getValue();
      if (valueElem == null) {
        provider.defaultSerializeNull(jgen);
      } else {
        try {
          valueSer.serialize(valueElem, jgen, provider);
        } catch (Exception e) {
          wrapAndThrow(provider, e, value, entry.getKey().name());
        }
      }
    }
  }

  //@Override
  public void resolve(SerializerProvider provider)
    throws JsonMappingException
  {
    if (_staticTyping) {
      _valueSerializer = provider.findValueSerializer(_valueType, _property);
    }
  }

  @SuppressWarnings("unchecked")
  @Override
  public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    throws JsonMappingException
  {
    ObjectNode o = createSchemaNode("object", true);
    if (typeHint instanceof ParameterizedType) {
      Type[] typeArgs = ((ParameterizedType) typeHint).getActualTypeArguments();
      if (typeArgs.length == 2) {
        JavaType enumType = TypeFactory.type(typeArgs[0]);
        JavaType valueType = TypeFactory.type(typeArgs[1]);
        ObjectNode propsNode = JsonNodeFactory.instance.objectNode();
```

```
            Class<Enum<?>> enumClass = (Class<Enum<?>>) enumType.getRawClass();
            for (Enum<?> enumValue : enumClass.getEnumConstants()) {
               JsonSerializer<Object> ser = provider.findValueSerializer(valueType.getRawClass(), _property);
               JsonNode schemaNode = (ser instanceof SchemaAware) ?
                     ((SchemaAware) ser).getSchema(provider, null) :
                     JsonSchema.getDefaultSchemaNode();
propsNode.put(provider.getConfig().getAnnotationIntrospector().findEnumValue((Enum<?>)enumValue), schemaN
ode);
            }
            o.put("properties", propsNode);
         }
      }
      return o;
   }
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.Type;

import org.codehaus.jackson.*;
import org.codehaus.jackson.io.SerializedString;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.introspect.BasicBeanDescription;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.map.util.EnumValues;
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.node.ArrayNode;
import org.codehaus.jackson.type.JavaType;

/**
* Standard serializer used for {@link java.lang.Enum} types.
*<p>
* Based on {@link ScalarSerializerBase} since the JSON value is
* scalar (String).
*
* @author tatu
*/
@JacksonStdImpl
public class EnumSerializer
   extends ScalarSerializerBase<Enum<?>>
{
   /**
    * This map contains pre-resolved values (since there are ways
    * to customize actual String constants to use) to use as
    * serializations.
    */
```

```
  protected final EnumValues _values;

  public EnumSerializer(EnumValues v) {
    super(Enum.class, false);
    _values = v;
  }

  public static EnumSerializer construct(Class<Enum<?>> enumClass, SerializationConfig config,
      BasicBeanDescription beanDesc)
  {
    // [JACKSON-212]: If toString() is to be used instead, leave EnumValues null
    AnnotationIntrospector intr = config.getAnnotationIntrospector();
    EnumValues v = config.isEnabled(SerializationConfig.Feature.WRITE_ENUMS_USING_TO_STRING)
      ? EnumValues.constructFromToString(enumClass, intr) : EnumValues.constructFromName(enumClass,
intr);
    return new EnumSerializer(v);
  }

  @Override
  public final void serialize(Enum<?> en, JsonGenerator jgen, SerializerProvider provider)
    throws IOException, JsonGenerationException
  {
    jgen.writeString(_values.serializedValueFor(en));
  }

  @Override
  public JsonNode getSchema(SerializerProvider provider, Type typeHint)
  {
    ObjectNode objectNode = createSchemaNode("string", true);
    if (typeHint != null) {
      JavaType type = TypeFactory.type(typeHint);
      if (type.isEnumType()) {
        ArrayNode enumNode = objectNode.putArray("enum");
        for (SerializedString value : _values.values()) {
          enumNode.add(value.getValue());
        }
      }
    }
    return objectNode;
  }

  public EnumValues getEnumValues() { return _values; }
}
package org.codehaus.jackson.map.ser;

import java.io.IOException;
import java.lang.reflect.Type;
```

```
import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.TypeSerializer;

public abstract class ScalarSerializerBase<T>
  extends SerializerBase<T>
{
  protected ScalarSerializerBase(Class<T> t) {
    super(t);
  }

  /**
   * Alternate constructor that is (alas!) needed to work
   * around kinks of generic type handling
   */
  @SuppressWarnings("unchecked")
  protected ScalarSerializerBase(Class<?> t, boolean dummy) {
    super((Class<T>) t);
  }

  /**
   * Default implementation will write type prefix, call regular serialization
   * method (since assumption is that value itself does not need JSON
   * Array or Object start/end markers), and then write type suffix.
   * This should work for most cases; some sub-classes may want to
   * change this behavior.
   */
  @Override
  public void serializeWithType(T value, JsonGenerator jgen, SerializerProvider provider,
      TypeSerializer typeSer)
    throws IOException, JsonGenerationException
  {
    typeSer.writeTypePrefixForScalar(value, jgen);
    serialize(value, jgen, provider);
    typeSer.writeTypeSuffixForScalar(value, jgen);
  }

  @Override
  public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    throws JsonMappingException
  {
    return createSchemaNode("string", true);
  }
}
```

```
package org.codehaus.jackson.map.ser;

import java.io.*;
import java.lang.reflect.Type;
import java.util.*;
import java.util.concurrent.atomic.*;

import org.codehaus.jackson.JsonGenerationException;
import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.util.Provider;

/**
* Class that providers access to serializers user for non-structured JDK types that
* are serializer as scalars; some using basic {@link ToStringSerializer},
* others explicit serializers.
*/
public class JdkSerializers
  implements Provider<Map.Entry<Class<?>,Object>>
{
  /**
   * Method called by {@link BasicSerializerFactory} to access
   * all serializers this class provides.
   */
  public Collection<Map.Entry<Class<?>, Object>> provide()
  {
     HashMap<Class<?>,Object> sers = new HashMap<Class<?>,Object>();

     // First things that 'toString()' can handle
     final ToStringSerializer sls = ToStringSerializer.instance;

     sers.put(java.net.URL.class, sls);
     sers.put(java.net.URI.class, sls);

     sers.put(Currency.class, sls);
     sers.put(UUID.class, sls);
     sers.put(java.util.regex.Pattern.class, sls);
     sers.put(Locale.class, sls);

     // starting with 1.7, use compact String for Locale
     sers.put(Locale.class, sls);

     // then atomic types
     sers.put(AtomicReference.class, AtomicReferenceSerializer.class);
     sers.put(AtomicBoolean.class, AtomicBooleanSerializer.class);
     sers.put(AtomicInteger.class, AtomicIntegerSerializer.class);
     sers.put(AtomicLong.class, AtomicLongSerializer.class);
```

```
    // then types that need specialized serializers
    sers.put(File.class, FileSerializer.class);
    sers.put(Class.class, ClassSerializer.class);

    // And then some stranger types... not 100% they are needed but:
    sers.put(Void.TYPE, NullSerializer.class);

    return sers.entrySet();
}

/*
 ******************************************************
 * Serializers for atomic types
 ******************************************************
 */

public final static class AtomicBooleanSerializer
    extends ScalarSerializerBase<AtomicBoolean>
{
    public AtomicBooleanSerializer() { super(AtomicBoolean.class, false); }

    @Override
    public void serialize(AtomicBoolean value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeBoolean(value.get());
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        return createSchemaNode("boolean", true);
    }
}

public final static class AtomicIntegerSerializer
    extends ScalarSerializerBase<AtomicInteger>
{
    public AtomicIntegerSerializer() { super(AtomicInteger.class, false); }

    @Override
    public void serialize(AtomicInteger value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeNumber(value.get());
    }
```

```
    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        return createSchemaNode("integer", true);
    }
}

public final static class AtomicLongSerializer
    extends ScalarSerializerBase<AtomicLong>
{
    public AtomicLongSerializer() { super(AtomicLong.class, false); }

    @Override
    public void serialize(AtomicLong value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        jgen.writeNumber(value.get());
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        return createSchemaNode("integer", true);
    }
}

public final static class AtomicReferenceSerializer
    extends SerializerBase<AtomicReference<?>>
{
    public AtomicReferenceSerializer() { super(AtomicReference.class, false); }

    @Override
    public void serialize(AtomicReference<?> value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
    {
        provider.defaultSerializeValue(value.get(), jgen);
    }

    @Override
    public JsonNode getSchema(SerializerProvider provider, Type typeHint)
    {
        return createSchemaNode("any", true);
    }
}

/*
/**********************************************************
/* Specialized serializers, referential types
```

```
/***********************************************************
 */

/**
 * For now, File objects get serialized by just outputting
 * absolute (but not canonical) name as String value
 */
public final static class FileSerializer
   extends ScalarSerializerBase<File>
{
   public FileSerializer() { super(File.class); }

   @Override
   public void serialize(File value, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonGenerationException
   {
      jgen.writeString(value.getAbsolutePath());
   }

   @Override
   public JsonNode getSchema(SerializerProvider provider, Type typeHint)
   {
      return createSchemaNode("string", true);
   }
}

/**
 * Also: default bean access will not do much good with Class.class. But
 * we can just serialize the class name and that should be enough.
 */
@SuppressWarnings("unchecked")
public final static class ClassSerializer
   extends ScalarSerializerBase<Class>
{
   public ClassSerializer() { super(Class.class); }

   @Override
   public void serialize(Class value, JsonGenerator jgen, SerializerProvider provider)
      throws IOException, JsonGenerationException
   {
      jgen.writeString(value.getName());
   }

   @Override
   public JsonNode getSchema(SerializerProvider provider, Type typeHint)
   {
      return createSchemaNode("string", true);
   }
```

```
  }

  /*
  /**********************************************************
  /* Other serializers
  /**********************************************************
   */

  /*
  public final static class LocaleSerializer
     extends ScalarSerializerBase<Locale>
  {
     public LocaleSerializer() { super(Locale.class); }

     @Override
     public void serialize(Locale value, JsonGenerator jgen, SerializerProvider provider)
        throws IOException, JsonGenerationException
     {
        jgen.writeString(value.toString());
     }

     @Override
     public JsonNode getSchema(SerializerProvider provider, Type typeHint)
        throws JsonMappingException
     {
        return createSchemaNode("string", true);
     }
  }
  */
}
package org.codehaus.jackson.map.ser;

import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.map.SerializerProvider;

/**
* Interface that defines API for filter objects use (as configured
* using {@link org.codehaus.jackson.map.annotate.JsonFilter})
* for filtering bean properties to serialize.
*
* @since 1.7
*/
public interface BeanPropertyFilter
{
  /**
   * Method called by {@link BeanSerializer} to let filter decide what to do with
   * given bean property value: the usual choices are to either filter out (i.e.
   * do nothing) or write using given {@link BeanPropertyWriter}, although filters
```

```
     * can choose other to do something different altogether.
     *
     * @param bean Bean of which property value to serialize
     * @param jgen Generator use for serializing value
     * @param prov Provider that can be used for accessing dynamic aspects of serialization
     *    processing
     * @param writer Default bean property serializer to use
     */
    public void serializeAsField(Object bean, JsonGenerator jgen, SerializerProvider prov,
            BeanPropertyWriter writer)
        throws Exception;
}
package org.codehaus.jackson.map;

import java.io.IOException;
import java.util.Date;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.ser.FilterProvider;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.schema.JsonSchema;
import org.codehaus.jackson.type.JavaType;

/**
* Abstract class that defines API used by {@link ObjectMapper} and
* {@link JsonSerializer}s to obtain serializers capable of serializing
* instances of specific types.
*<p>
* Note about usage: for {@link JsonSerializer} instances, only accessors
* for locating other (sub-)serializers are to be used. {@link ObjectMapper},
* on the other hand, is to initialize recursive serialization process by
* calling {@link #serializeValue}.
*/
public abstract class SerializerProvider
{
    protected final static JavaType TYPE_OBJECT = TypeFactory.type(Object.class);

    /**
     * Serialization configuration to use for serialization processing.
     */
    protected final SerializationConfig _config;

    /**
     * View used for currently active serialization
     */
    protected final Class<?> _serializationView;

    protected SerializerProvider(SerializationConfig config)
```

```
{
   _config = config;
   _serializationView = (config == null) ? null : _config.getSerializationView();
}

/*
/**********************************************************
/* Methods that ObjectMapper will call
/**********************************************************
 */

/**
 * The method to be called by {@link ObjectMapper} to
 * execute recursive serialization, using serializers that
 * this provider has access to.
 *
 * @param jsf Underlying factory object used for creating serializers
 *    as needed
 */
public abstract void serializeValue(SerializationConfig cfg, JsonGenerator jgen,
       Object value, SerializerFactory jsf)
   throws IOException, JsonGenerationException;

/**
 * The method to be called by {@link ObjectMapper} to
 * execute recursive serialization, using serializers that
 * this provider has access to; and using specified root type
 * for locating first-level serializer.
 *
 * @param rootType Type to use for locating serializer to use, instead of actual
 *    runtime type. Must be actual type, or one of its super types
 *
 * @since 1.5
 */
public abstract void serializeValue(SerializationConfig cfg, JsonGenerator jgen,
       Object value, JavaType rootType, SerializerFactory jsf)
   throws IOException, JsonGenerationException;

/**
 * Generate <a href="http://json-schema.org/">Json-schema</a> for
 * given type.
 *
 * @param type The type for which to generate schema
 */
public abstract JsonSchema generateJsonSchema(Class<?> type, SerializationConfig config, SerializerFactory jsf)
   throws JsonMappingException;

/**
```

```
     * Method that can be called to see if this serializer provider
     * can find a serializer for an instance of given class.
     *<p>
     * Note that no Exceptions are thrown, including unchecked ones:
     * implementations are to swallow exceptions if necessary.
     */
    public abstract boolean hasSerializerFor(SerializationConfig cfg,
                                             Class<?> cls, SerializerFactory jsf);

    /*
    /**********************************************************
    /* Access to configuration
    /**********************************************************
     */

    /**
     * Method for accessing configuration for the serialization processing.
     */
    public final SerializationConfig getConfig() { return _config; }

    /**
     * Convenience method for checking whether specified serialization
     * feature is enabled or not.
     * Shortcut for:
     *<pre>
     *  getConfig().isEnabled(feature);
     *</pre>
     */
    public final boolean isEnabled(SerializationConfig.Feature feature) {
       return _config.isEnabled(feature);
    }

    /**
     * Convenience method for accessing serialization view in use (if any); equivalent to:
     *<pre>
     *   getConfig().getSerializationView();
     *</pre>
     *
     * @since 1.4
     */
    public final Class<?> getSerializationView() { return _serializationView; }

    /**
     * Convenience method for accessing provider to find serialization filters used,
     * equivalent to calling:
     *<pre>
     *   getConfig().getFilterProvider();
     *</pre>
```

```
     *
     * @since 1.4
     */
    public final FilterProvider getFilterProvider() {
        return _config.getFilterProvider();
    }

    /*
    /**********************************************************
    /* General serializer locating functionality
    /**********************************************************
     */

    /**
     * Method called to get hold of a serializer for a value of given type;
     * or if no such serializer can be found, a default handler (which
     * may do a best-effort generic serialization or just simply
     * throw an exception when invoked).
     *<p>
     * Note: this method is only called for non-null values; not for keys
     * or null values. For these, check out other accessor methods.
     *<p>
     * Note that starting with version 1.5, serializers should also be type-aware
     * if they handle polymorphic types. That means that it may be necessary
     * to also use a {@link TypeSerializer} based on declared (static) type
     * being serializer (whereas actual data may be serialized using dynamic
     * type)
     *
     * @throws JsonMappingException if there are fatal problems with
     *   accessing suitable serializer; including that of not
     *   finding any serializer
     */
    public abstract JsonSerializer<Object> findValueSerializer(Class<?> runtimeType,
            BeanProperty property)
        throws JsonMappingException;

    /**
     * Similar to {@link #findValueSerializer(Class)}, but takes full generics-aware
     * type instead of raw class.
     *
     * @since 1.5
     */
    public abstract JsonSerializer<Object> findValueSerializer(JavaType serializationType,
            BeanProperty property)
        throws JsonMappingException;

    /**
     * Method called to locate regular serializer, matching type serializer,
```

```
 * and if both found, wrap them in a serializer that calls both in correct
 * sequence. This method is currently only used for root-level serializer
 * handling to allow for simpler caching. A call can always be replaced
 * by equivalent calls to access serializer and type serializer separately.
 *
 * @param valueType Type for purpose of locating a serializer; usually dynamic
 *   runtime type, but can also be static declared type, depending on configuration
 *
 * @param cache Whether resulting value serializer should be cached or not; this is just
 *    a hint
 *
 * @since 1.5
 */
public abstract JsonSerializer<Object> findTypedValueSerializer(Class<?> valueType,
       boolean cache, BeanProperty property)
   throws JsonMappingException;

/**
 * Method called to locate regular serializer, matching type serializer,
 * and if both found, wrap them in a serializer that calls both in correct
 * sequence. This method is currently only used for root-level serializer
 * handling to allow for simpler caching. A call can always be replaced
 * by equivalent calls to access serializer and type serializer separately.
 *
 * @param valueType Declared type of value being serialized (which may not
 *    be actual runtime type); used for finding both value serializer and
 *    type serializer to use for adding polymorphic type (if any)
 *
 * @param cache Whether resulting value serializer should be cached or not; this is just
 *    a hint
 *
 * @since 1.5
 */
public abstract JsonSerializer<Object> findTypedValueSerializer(JavaType valueType,
       boolean cache, BeanProperty property)
   throws JsonMappingException;

/**
 * Method called to get the serializer to use for serializing
 * non-null Map keys. Separation from regular
 * {@link #findValueSerializer} method is because actual write
 * method must be different (@link JsonGenerator#writeFieldName};
 * but also since behavior for some key types may differ.
 *<p>
 * Note that the serializer itself can be called with instances
 * of any Java object, but not nulls.
 */
public abstract JsonSerializer<Object> getKeySerializer(JavaType keyType,
```

```
        BeanProperty property)
    throws JsonMappingException;

/*
/**********************************************************
/* Deprecated serializer locating functionality (as of 1.7)
/**********************************************************
 */

/**
 * Deprecated version of accessor method that was used before version 1.7.
 * Implemented as final to ensure that existing code does not accidentally
 * try to redefine it (given that it is not called by core mapper code)
 *
 * @deprecated As of version 1.7, use version that exposes property object
 *    instead of just its type (needed for contextual serializers)
 */
@Deprecated
public final JsonSerializer<Object> findValueSerializer(Class<?> runtimeType)
    throws JsonMappingException
{
    return findValueSerializer(runtimeType, null);
}

/**
 * Deprecated version of accessor method that was used before version 1.7.
 * Implemented as final to ensure that existing code does not accidentally
 * try to redefine it (given that it is not called by core mapper code)
 *
 * @deprecated As of version 1.7, use version that exposes property object
 *    instead of just its type (needed for contextual serializers)
 */
@Deprecated
public final JsonSerializer<Object> findValueSerializer(JavaType serializationType)
    throws JsonMappingException
{
    return findValueSerializer(serializationType, null);
}

/**
 * Deprecated version of accessor method that was used before version 1.7.
 * Implemented as final to ensure that existing code does not accidentally
 * try to redefine it (given that it is not called by core mapper code)
 *
 * @deprecated As of version 1.7, use version that exposes property object
 *    instead of just its type (needed for contextual serializers)
 */
@Deprecated
```

```
public final JsonSerializer<Object> findTypedValueSerializer(Class<?> valueType,
      boolean cache)
   throws JsonMappingException
{
   return findTypedValueSerializer(valueType, cache, null);
}

/**
 * Deprecated version of accessor method that was used before version 1.7.
 * Implemented as final to ensure that existing code does not accidentally
 * try to redefine it (given that it is not called by core mapper code)
 *
 * @deprecated As of version 1.7, use version that exposes property object
 *    instead of just its type (needed for contextual serializers)
 */
@Deprecated
public final JsonSerializer<Object> findTypedValueSerializer(JavaType valueType,
      boolean cache)
   throws JsonMappingException
{
   return findTypedValueSerializer(valueType, cache, null);
}

/**
 * Deprecated version of accessor method that was used before version 1.7.
 * Implemented as final to ensure that existing code does not accidentally
 * try to redefine it (given that it is not called by core mapper code)
 *
 * @deprecated As of version 1.7, use version that exposes property object
 *    instead of just its type (needed for contextual serializers)
 */
@Deprecated
public final JsonSerializer<Object> getKeySerializer()
   throws JsonMappingException
{
   return getKeySerializer(TYPE_OBJECT, null);
}

/*
/**********************************************************
/* Accessors for specialized serializers
/**********************************************************
 */

/**
 * Method called to get the serializer to use for serializing
 * Map keys that are nulls: this is needed since JSON does not allow
 * any non-String value as key, including null.
```

```
     *<p>
     * Typically, returned serializer
     * will either throw an exception, or use an empty String; but
     * other behaviors are possible.
     */
    public abstract JsonSerializer<Object> getNullKeySerializer();

    /**
     * Method called to get the serializer to use for serializing
     * values (root level, Array members or List field values)
     * that are nulls. Specific accessor is needed because nulls
     * in Java do not contain type information.
     *<p>
     * Typically returned serializer just writes out Json literal
     * null value.
     */
    public abstract JsonSerializer<Object> getNullValueSerializer();

    /**
     * Method called to get the serializer to use if provider
     * can not determine an actual type-specific serializer
     * to use; typically when none of {@link SerializerFactory}
     * instances are able to construct a serializer.
     *<p>
     * Typically, returned serializer will throw an exception,
     * although alternatively {@link org.codehaus.jackson.map.ser.ToStringSerializer} could
     * be returned as well.
     *
     * @param unknownType Type for which no serializer is found
     */
    public abstract JsonSerializer<Object> getUnknownTypeSerializer(Class<?> unknownType);

    /*
    /**********************************************************
    /* Convenience methods
    /**********************************************************
     */

    /**
     * Convenience method that will serialize given value (which can be
     * null) using standard serializer locating functionality. It can
     * be called for all values including field and Map values, but usually
     * field values are best handled calling
     * {@link #defaultSerializeField} instead.
     */
    public final void defaultSerializeValue(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException
    {
```

```
    if (value == null) {
        getNullValueSerializer().serialize(null, jgen, this);
    } else {
        Class<?> cls = value.getClass();
        findTypedValueSerializer(cls, true).serialize(value, jgen, this);
    }
}

/**
 * Convenience method that will serialize given field with specified
 * value. Value may be null. Serializer is done using the usual
 * null) using standard serializer locating functionality.
 */
public final void defaultSerializeField(String fieldName, Object value, JsonGenerator jgen)
    throws IOException, JsonProcessingException
{
    jgen.writeFieldName(fieldName);
    if (value == null) {
        /* Note: can't easily check for suppression at this point
         * any more; caller must check it.
         */
        getNullValueSerializer().serialize(null, jgen, this);
    } else {
        Class<?> cls = value.getClass();
        findTypedValueSerializer(cls, true).serialize(value, jgen, this);
    }
}

/**
 * Method that will handle serialization of Date(-like) values, using
 * {@link SerializationConfig} settings to determine expected serialization
 * behavior.
 * Note: date here means "full" date, that is, date AND time, as per
 * Java convention (and not date-only values like in SQL)
 */
public abstract void defaultSerializeDateValue(long timestamp, JsonGenerator jgen)
    throws IOException, JsonProcessingException;

/**
 * Method that will handle serialization of Date(-like) values, using
 * {@link SerializationConfig} settings to determine expected serialization
 * behavior.
 * Note: date here means "full" date, that is, date AND time, as per
 * Java convention (and not date-only values like in SQL)
 */
public abstract void defaultSerializeDateValue(Date date, JsonGenerator jgen)
    throws IOException, JsonProcessingException;
```

```
  /**
   * @since 1.7
   */
  public final void defaultSerializeNull(JsonGenerator jgen)
     throws IOException, JsonProcessingException
  {
     getNullValueSerializer().serialize(null, jgen, this);
  }

  /*
  /**********************************************************
  /* Access to caching details
  /**********************************************************
   */

  /**
   * Method that can be used to determine how many serializers this
   * provider is caching currently
   * (if it does caching: default implementation does)
   * Exact count depends on what kind of serializers get cached;
   * default implementation caches all serializers, including ones that
   * are eagerly constructed (for optimal access speed)
   *<p>
   * The main use case for this method is to allow conditional flushing of
   * serializer cache, if certain number of entries is reached.
   *
   * @since 1.4
   */
  public abstract int cachedSerializersCount();

  /**
   * Method that will drop all serializers currently cached by this provider.
   * This can be used to remove memory usage (in case some serializers are
   * only used once or so), or to force re-construction of serializers after
   * configuration changes for mapper than owns the provider.
   *
   * @since 1.4
   */
  public abstract void flushCachedSerializers();

}
package org.codehaus.jackson.map;

import java.io.IOException;

import org.codehaus.jackson.*;
```

```
/**
* Abstract class that defines API used for deserializing Json content
* field names into Java Map keys. These deserializers are only used
* if the Map key class is not <code>String</code> or <code>Object</code>.
*/
public abstract class KeyDeserializer
{
  public abstract Object deserializeKey(String key, DeserializationContext ctxt)
     throws IOException, JsonProcessingException;

  /**
   * This marker class is only to be used with annotations, to
   * indicate that <b>no deserializer is configured</b>.
   *<p>
   * Specifically, this class is to be used as the marker for
   * annotation {@link org.codehaus.jackson.map.annotate.JsonDeserialize}.
   *
   * @since 1.3
   */
  public abstract static class None
     extends KeyDeserializer { }
}
/**
* Package that contains classes and interfaces to help implement
* custom extension {@link org.codehaus.jackson.map.Module}s
* (which are registered using
* {@link org.codehaus.jackson.map.ObjectMapper#registerModule}.
*
* @since 1.7
*/
package org.codehaus.jackson.map.module;
package org.codehaus.jackson.map.module;

import java.util.*;

import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.type.ClassKey;
import org.codehaus.jackson.type.JavaType;

/**
* Simple implementation {@link Serializers} which allows registration of
* serializers based on raw (type erased class).
* It can work well for basic bean and scalar type serializers, but is not
* a good fit for handling generic types (like {@link Map}s and {@link Collection}s).
*<p>
* Type registrations are assumed to be general; meaning that registration of serializer
* for a super type will also be used for handling subtypes, unless an exact match
* is found first. As an example, handler for {@link CharSequence} would also be used
```

```
* serializing {@link StringBuilder} instances, unless a direct mapping was found.
*
* @since 1.7
*/
public class SimpleSerializers implements Serializers
{
  /**
   * Class-based mappings that are used both for exact and
   * sub-class matches.
   */
  protected HashMap<ClassKey,JsonSerializer<?>> _classMappings = null;

  /**
   * Interface-based matches.
   */
  protected HashMap<ClassKey,JsonSerializer<?>> _interfaceMappings = null;

  /*
  /**********************************************************
  /* Life-cycle, construction and configuring
  /**********************************************************
   */

  public SimpleSerializers() { }

  /**
   * Method for adding given serializer for type that {@link JsonSerializer#handledType}
   * specifies (which MUST return a non-null class; and can NOT be {@link Object}, as a
   * sanity check).
   * For serializers that do not declare handled type, use the variant that takes
   * two arguments.
   *
   * @param ser
   */
  public void addSerializer(JsonSerializer<?> ser)
  {
     // Interface to match?
     Class<?> cls = ser.handledType();
     if (cls == null || cls == Object.class) {
        throw new IllegalArgumentException("JsonSerializer of type "+ser.getClass().getName()
              +" does not define valid handledType() (use alternative registration method?)");
     }
     _addSerializer(cls, ser);
  }

  public <T> void addSerializer(Class<? extends T> type, JsonSerializer<T> ser)
  {
     _addSerializer(type, ser);
```

```
}

private void _addSerializer(Class<?> cls, JsonSerializer<?> ser)
{
   ClassKey key = new ClassKey(cls);
   // Interface or class type?
   if (cls.isInterface()) {
      if (_interfaceMappings == null) {
         _interfaceMappings = new HashMap<ClassKey,JsonSerializer<?>>();
      }
      _interfaceMappings.put(key, ser);
   } else { // nope, class:
      if (_classMappings == null) {
         _classMappings = new HashMap<ClassKey,JsonSerializer<?>>();
      }
      _classMappings.put(key, ser);
   }
}

/*
/**********************************************************
/* Serializers implementation
/**********************************************************
 */

@Override
public JsonSerializer<?> findSerializer(SerializationConfig config, JavaType type,
       BeanDescription beanDesc, BeanProperty property)
{
   Class<?> cls = type.getRawClass();
   ClassKey key = new ClassKey(cls);
   JsonSerializer<?> ser = null;

   // First: direct match?
   if (cls.isInterface()) {
      if (_interfaceMappings != null) {
         ser = _interfaceMappings.get(key);
         if (ser != null) {
            return ser;
         }
      }
   } else {
      if (_classMappings != null) {
         ser = _classMappings.get(key);
         if (ser != null) {
            return ser;
         }
         // If not direct match, maybe super-class match?
```

```
            for (Class<?> curr = cls; (curr != null); curr = curr.getSuperclass()) {
                key.reset(curr);
                ser = _classMappings.get(key);
                if (ser != null) {
                    return ser;
                }
            }
        }
    }
    // No direct match? How about super-interfaces?
    if (_interfaceMappings != null) {
        return _findInterfaceMapping(cls, key);
    }
    return null;
  }

  /*
  /**********************************************************
  /* Internal methods
  /**********************************************************
   */

  protected JsonSerializer<?> _findInterfaceMapping(Class<?> cls, ClassKey key)
  {
    for (Class<?> iface : cls.getInterfaces()) {
        key.reset(iface);
        JsonSerializer<?> ser = _interfaceMappings.get(key);
        if (ser != null) {
            return ser;
        }
        ser = _findInterfaceMapping(iface, key);
        if (ser != null) {
            return ser;
        }
    }
    return null;
  }
}
package org.codehaus.jackson.map.module;

import java.util.*;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.type.ArrayType;
import org.codehaus.jackson.map.type.ClassKey;
import org.codehaus.jackson.map.type.CollectionType;
import org.codehaus.jackson.map.type.MapType;
```

```
import org.codehaus.jackson.type.JavaType;

/**
* Simple implementation {@link Deserializers} which allows registration of
* deserializers based on raw (type erased class).
* It can work well for basic bean and scalar type deserializers, but is not
* a good fit for handling generic types (like {@link Map}s and {@link Collection}s
* or array types).
*<p>
* Unlike {@link SimpleSerializers}, this class does not currently support generic mappings;
* all mappings must be to exact declared deserialization type.
*
* @since 1.7
*/
public class SimpleDeserializers implements Deserializers
{
  protected HashMap<ClassKey,JsonDeserializer<?>> _classMappings = null;

  /*
  /**********************************************************
  /* Life-cycle, construction and configuring
  /**********************************************************
   */

  public SimpleDeserializers() { }

  public <T> void addDeserializer(Class<T> forClass, JsonDeserializer<? extends T> deser)
  {
     ClassKey key = new ClassKey(forClass);
     if (_classMappings == null) {
        _classMappings = new HashMap<ClassKey,JsonDeserializer<?>>();
     }
     _classMappings.put(key, deser);
  }

  /*
  /**********************************************************
  /* Serializers implementation
  /**********************************************************
   */

  @Override
  public JsonDeserializer<?> findArrayDeserializer(ArrayType type,
        DeserializationConfig config, DeserializerProvider provider,
        BeanProperty property,
        TypeDeserializer elementTypeDeserializer,
        JsonDeserializer<?> elementDeserializer)
  {
```

```
    return (_classMappings == null) ? null : _classMappings.get(new ClassKey(type.getRawClass()));
  }

  @Override
  public JsonDeserializer<?> findBeanDeserializer(JavaType type,
      DeserializationConfig config, DeserializerProvider provider,
      BeanDescription beanDesc, BeanProperty property)
  {
    return (_classMappings == null) ? null : _classMappings.get(new ClassKey(type.getRawClass()));
  }

  @Override
  public JsonDeserializer<?> findCollectionDeserializer(CollectionType type,
      DeserializationConfig config, DeserializerProvider provider,
      BeanDescription beanDesc, BeanProperty property,
      TypeDeserializer elementTypeDeserializer,
      JsonDeserializer<?> elementDeserializer)
  {
    return (_classMappings == null) ? null : _classMappings.get(new ClassKey(type.getRawClass()));
  }

  @Override
  public JsonDeserializer<?> findEnumDeserializer(Class<?> type,
      DeserializationConfig config, BeanDescription beanDesc, BeanProperty property)
  {
    return (_classMappings == null) ? null : _classMappings.get(new ClassKey(type));
  }

  @Override
  public JsonDeserializer<?> findMapDeserializer(MapType type,
      DeserializationConfig config, DeserializerProvider provider,
      BeanDescription beanDesc, BeanProperty property,
      KeyDeserializer keyDeserializer,
      TypeDeserializer elementTypeDeserializer,
      JsonDeserializer<?> elementDeserializer)
  {
    return (_classMappings == null) ? null : _classMappings.get(new ClassKey(type.getRawClass()));
  }

  @Override
  public JsonDeserializer<?> findTreeNodeDeserializer(Class<? extends JsonNode> nodeType,
      DeserializationConfig config, BeanProperty property)
  {
    return (_classMappings == null) ? null : _classMappings.get(new ClassKey(nodeType));
  }
}
package org.codehaus.jackson.map.module;
```

```
import org.codehaus.jackson.Version;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.Module;

/**
* Simple {@link Module} implementation that allows registration
* of serializers and deserializers, and bean serializer
* and deserializer modifiers.
*
* @since 1.7
*/
public class SimpleModule extends Module
{
  protected final String _name;
  protected final Version _version;

  protected SimpleSerializers _serializers = null;
  protected SimpleDeserializers _deserializers = null;

  /*
  /**********************************************************
  /* Life-cycle: create, configure
  /**********************************************************
   */

  public SimpleModule(String name, Version version)
  {
     _name = name;
     _version = version;
  }

  public SimpleModule addSerializer(JsonSerializer<?> ser)
  {
     if (_serializers == null) {
        _serializers = new SimpleSerializers();
     }
     _serializers.addSerializer(ser);
     return this;
  }

  public <T> SimpleModule addSerializer(Class<? extends T> type, JsonSerializer<T> ser)
  {
     if (_serializers == null) {
        _serializers = new SimpleSerializers();
     }
     _serializers.addSerializer(type, ser);
     return this;
```

```
    }

    public <T> SimpleModule addDeserializer(Class<T> type, JsonDeserializer<? extends T> deser)
    {
        if (_deserializers == null) {
            _deserializers = new SimpleDeserializers();
        }
        _deserializers.addDeserializer(type, deser);
        return this;
    }

    /*
    /**********************************************************
    /* Module impl
    /**********************************************************
     */

    @Override
    public String getModuleName() {
        return _name;
    }

    @Override
    public void setupModule(SetupContext context)
    {
        if (_serializers != null) {
            context.addSerializers(_serializers);
        }
        if (_deserializers != null) {
            context.addDeserializers(_deserializers);
        }
    }

    @Override
    public Version version() {
        return _version;
    }
}
package org.codehaus.jackson.map;

import java.text.DateFormat;
import java.util.Map;

import org.codehaus.jackson.annotate.JsonAutoDetect;
import org.codehaus.jackson.map.introspect.VisibilityChecker;
import org.codehaus.jackson.map.jsontype.SubtypeResolver;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.type.JavaType;
```

```
/**
* Interface that defines functionality accessible through both
* serialization and deserialization configuration objects;
* accessors to mode-independent configuration settings
* and such.
*
* @since 1.2
*/
public interface MapperConfig<T extends MapperConfig<T>>
  extends ClassIntrospector.MixInResolver
{
  // // // Accessors

  // // // Life-cycle methods

  /**
   * Method that checks class annotations that the argument Object has,
   * and modifies settings of this configuration object accordingly,
   * similar to how those annotations would affect actual value classes
   * annotated with them, but with global scope. Note that not all
   * annotations have global significance, and thus only subset of
   * Jackson annotations will have any effect.
   */
  public void fromAnnotations(Class<?> cls);

  /**
   * Method to use for constructing an instance that is not shared
   * between multiple operations but only used for a single one.
   */
  public T createUnshared(TypeResolverBuilder<?> typer, VisibilityChecker<?> vc,
        SubtypeResolver subtypeResolver);

  // // // Configuration

  public AnnotationIntrospector getAnnotationIntrospector();

  /**
   * Method for replacing existing annotation introspector(s) with specified
   * introspector.
   */
  public void setAnnotationIntrospector(AnnotationIntrospector introspector);

  /**
   * Method for registering specified {@link AnnotationIntrospector} as the highest
   * priority introspector (will be chained with existing introspector(s) which
   * will be used as fallbacks for cases this introspector does not handle)
   *
```

```
     * @param introspector Annotation introspector to register.
     *
     * @since 1.7
     */
    public void insertAnnotationIntrospector(AnnotationIntrospector introspector);

    /**
     * Method for registering specified {@link AnnotationIntrospector} as the lowest
     * priority introspector, chained with existing introspector(s) and called
     * as fallback for cases not otherwise handled.
     *
     * @param ai Annotation introspector to register.
     *
     * @since 1.7
     */
    public void appendAnnotationIntrospector(AnnotationIntrospector ai);

    /**
     * Method for replacing existing {@link ClassIntrospector} with
     * specified replacement.
     */
    public void setIntrospector(ClassIntrospector<? extends BeanDescription> i);

    /**
     * Method to use for defining mix-in annotations to use for augmenting
     * annotations that processable (serializable / deserializable)
     * classes have.
     * Mixing in is done when introspecting class annotations and properties.
     * Map passed contains keys that are target classes (ones to augment
     * with new annotation overrides), and values that are source classes
     * (have annotations to use for augmentation).
     * Annotations from source classes (and their supertypes)
     * will <b>override</b>
     * annotations that target classes (and their super-types) have.
     *
     * @since 1.2
     */
    public void setMixInAnnotations(Map<Class<?>, Class<?>> mixins);

    /**
     * Method to use for adding mix-in annotations to use for augmenting
     * specified class or interface. All annotations from
     * <code>mixinSource</code> are taken to override annotations
     * that <code>target</code> (or its supertypes) has.
     *
     * @since 1.2
     *
     * @param target Class (or interface) whose annotations to effectively override
```

```
     * @param mixinSource Class (or interface) whose annotations are to
     *   be "added" to target's annotations, overriding as necessary
     */
    public void addMixInAnnotations(Class<?> target, Class<?> mixinSource);

    // ClassIntrospector.MixInResolver impl:

    /**
     * Method that will check if there are "mix-in" classes (with mix-in
     * annotations) for given class
     */
    public Class<?> findMixInClassFor(Class<?> cls);

    /**
     * Method for accessing currently configured (textual) date format
     * that will be used for reading or writing date values (in case
     * of writing, only if textual output is configured; not if dates
     * are to be serialized as time stamps).
     *<p>
     * Note that typically {@link DateFormat} instances are <b>not thread-safe</b>
     * (at least ones provided by JDK):
     * this means that calling code should clone format instance before
     * using it.
     *<p>
     * This method is usually only called by framework itself, since there
     * are convenience methods available via
     * {@link DeserializationContext} and {@link SerializerProvider} that
     * take care of cloning and thread-safe reuse.
     */
    public DateFormat getDateFormat();

    /**
     * Method that wiTll define specific date format to use for reading/writing
     * Date and Calendar values; instance is used as is, without creating
     * a clone.
     * Format object can be access using
     * {@link #getDateFormat}.
     */
    public void setDateFormat(DateFormat df);

    /**
     * Method called to locate a type info handler for types that do not have
     * one explicitly declared via annotations (or other configuration).
     * If such default handler is configured, it is returned; otherwise
     * null is returned.
     *
     * @since 1.5
     */
```

```
  public TypeResolverBuilder<?> getDefaultTyper(JavaType baseType);

  /**
   * Accessor for object used for determining whether specific property elements
   * (method, constructors, fields) can be auto-detected based on
   * their visibility (access modifiers). Can be changed to allow
   * different minimum visibility levels for auto-detection. Note
   * that this is the global handler; individual types (classes)
   * can further override active checker used (using
   * {@link JsonAutoDetect} annotation)
   *
   * @since 1.5
   */
  public VisibilityChecker<?> getDefaultVisibilityChecker();

  /**
   * Accessor for object used for finding out all reachable subtypes
   * for supertypes; needed when a logical type name is used instead
   * of class name (or custom scheme).
   *
   * @since 1.6
   */
  public SubtypeResolver getSubtypeResolver();

  /**
   * Method for overriding subtype resolver used.
   *
   * @since 1.6
   */
  public void setSubtypeResolver(SubtypeResolver r);

  /**
   * Accessor for getting bean description that only contains class
   * annotations: useful if no getter/setter/creator information is needed.
   *
   * @since 1.7
   */
  public <DESC extends BeanDescription> DESC introspectClassAnnotations(Class<?> cls);

  /**
   * Accessor for getting bean description that only contains immediate class
   * annotations: ones from the class, and its direct mix-in, if any, but
   * not from super types.
   */
  public <DESC extends BeanDescription> DESC introspectDirectClassAnnotations(Class<?> cls);
}
package org.codehaus.jackson.map;
```

```
import java.io.IOException;

import org.codehaus.jackson.*;

/**
* Interface that can be implemented by objects that know how to
* serialize themselves to JSON, using {@link JsonGenerator}
* (and {@link SerializerProvider} if necessary).
*<p>
* Note that implementing this interface binds implementing object
* closely to Jackson API, and that it is often not necessary to do
* so -- if class is a bean, it can be serialized without
* implementing this interface.
*<p>
* NOTE: as of version 1.5, this interface is missing one crucial
* aspect, that of dealing with type information embedding.
* Because of this, this interface is deprecated, although will be
* fully supported for all 1.x releases, and will work except for
* cases where polymorphic type information handling is needed for
* type (in which case implementing if {@link JsonSerializableWithType} is crucial).
*
* @see org.codehaus.jackson.map.JsonSerializableWithType
*
* @since 1.5
* @deprecated Use {@link JsonSerializableWithType} instead
*/
@Deprecated
public interface JsonSerializable
{
  public void serialize(JsonGenerator jgen, SerializerProvider provider)
     throws IOException, JsonProcessingException;
}
package org.codehaus.jackson.map;

import java.lang.annotation.Annotation;

import org.codehaus.jackson.map.introspect.AnnotatedMember;
import org.codehaus.jackson.map.util.Annotations;
import org.codehaus.jackson.type.JavaType;

/**
* Bean properties are logical entities that represent data
* Java objects ("beans", although more accurately POJOs)
* contain; and that are accessed using some combination
* of methods (getter, setter), field and constructor
* parameter.
*
* @since 1.7
```

```
*/
public interface BeanProperty
{
  /**
   * Method to get logical name of the property
   */
  public String getName();

  /**
   * Method to get declared type of the property.
   */
  public JavaType getType();

  /**
   * Method for finding annotation associated with this property;
   * meaning annotation associated with one of entities used to
   * access property.
   */
  public <A extends Annotation> A getAnnotation(Class<A> acls);

  /**
   * Method for finding annotation associated with context of
   * this property; usually class in which member is declared
   * (or its subtype if processing subtype).
   */
  public <A extends Annotation> A getContextAnnotation(Class<A> acls);

  /**
   * Method for accessing primary physical entity that represents the property;
   * annotated field, method or constructor property.
   */
  public AnnotatedMember getMember();

  /*
  /**********************************************************
  /* Simple stand-alone implementation, useful as a placeholder
  /**********************************************************
   */

  public static class Std implements BeanProperty
  {
     protected final String _name;
     protected final JavaType _type;

     /**
      * Physical entity (field, method or constructor argument) that
      * is used to access value of property (or in case of constructor
      * property, just placeholder)
```

```
     */
    protected final AnnotatedMember _member;

    /**
     * Annotations defined in the context class (if any); may be null
     * if no annotations were found
     */
    protected final Annotations _contextAnnotations;

    public Std(String name, JavaType type, Annotations contextAnnotations, AnnotatedMember member)
    {
        _name = name;
        _type = type;
        _member = member;
        _contextAnnotations = contextAnnotations;
    }

    public Std withType(JavaType type) {
        return new Std(_name, type, _contextAnnotations, _member);
    }

    public <A extends Annotation> A getAnnotation(Class<A> acls) {
        return _member.getAnnotation(acls);
    }

    public <A extends Annotation> A getContextAnnotation(Class<A> acls) {
        return (_contextAnnotations == null) ? null : _contextAnnotations.get(acls);
    }

    public String getName() {
        return _name;
    }

    public JavaType getType() {
        return _type;
    }

    public AnnotatedMember getMember() {
        return _member;
    }
  }
}
package org.codehaus.jackson.map;

import org.codehaus.jackson.type.JavaType;

/**
* Helper class used to introspect features of POJO value classes
```

```
* used with Jackson. The main use is for finding out
* POJO construction (creator) and value access (getters, setters)
* methods and annotations that define configuration of using
* those methods.
*/
public abstract class ClassIntrospector<T extends BeanDescription>
{
  /*
  //////////////////////////////////////////////////////
  // Helper interfaces
  //////////////////////////////////////////////////////
   */

  /**
   * Interface used for decoupling details of how mix-in annotation
   * definitions are accessed (via this interface), and how
   * they are stored (defined by classes that implement the interface)
   */
  public interface MixInResolver
  {
     /**
      * Method that will check if there are "mix-in" classes (with mix-in
      * annotations) for given class
      */
     public Class<?> findMixInClassFor(Class<?> cls);
  }

  protected ClassIntrospector() { }

  /*
  //////////////////////////////////////////////////////
  // Public API: factory methods
  //////////////////////////////////////////////////////
   */

  /**
   * Factory method that constructs an introspector that has all
   * information needed for serialization purposes.
   */
  public abstract T forSerialization(SerializationConfig cfg, JavaType type,
                                     MixInResolver r);

  /**
   * Factory method that constructs an introspector that has all
   * information needed for deserialization purposes.
   */
  public abstract T forDeserialization(DeserializationConfig cfg, JavaType type,
                                       MixInResolver r);
```

```
  /**
   * Factory method that constructs an introspector that has
   * information necessary for creating instances of given
   * class ("creator"), as well as class annotations, but
   * no information on member methods
   */
  public abstract T forCreation(DeserializationConfig cfg, JavaType type,
                                MixInResolver r);

  /**
   * Factory method that constructs an introspector that only has
   * information regarding annotations class itself (or its supertypes) has,
   * but nothing on methods or constructors.
   */
  public abstract T forClassAnnotations(MapperConfig<?> cfg, Class<?> c,
                                        MixInResolver r);

  /**
   * Factory method that constructs an introspector that only has
   * information regarding annotations class itself has (but NOT including
   * its supertypes), but nothing on methods or constructors.
   *
   * @since 1.5
   */
  public abstract T forDirectClassAnnotations(MapperConfig<?> cfg, Class<?> c,
        MixInResolver r);

}
package org.codehaus.jackson.map;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.deser.BeanDeserializerModifier;
import org.codehaus.jackson.map.type.*;
import org.codehaus.jackson.type.JavaType;

/**
* Abstract class that defines API used by {@link DeserializerProvider}
* to obtain actual
* {@link JsonDeserializer} instances from multiple distinct factories.
*<p>
* Since there are multiple broad categories of deserializers, there are
* multiple factory methods:
*<ul>
* <li>For JSON "Array" type, we need 2 methods: one to deal with expected
*   Java arrays ({@link #createArrayDeserializer})
*   and the other for other Java containers like {@link java.util.List}s
*   and {@link java.util.Set}s ({@link #createCollectionDeserializer(DeserializationConfig, DeserializerProvider,
```

```
CollectionType, BeanProperty)})
*  </li>
* <li>For JSON "Object" type, we need 2 methods: one to deal with
*   expected Java {@link java.util.Map}s
*   ({@link #createMapDeserializer}), and another for POJOs
*   ({@link #createBeanDeserializer(DeserializationConfig, DeserializerProvider, JavaType, BeanProperty)}.
*  </li>
* <li>For Tree Model ({@link org.codehaus.jackson.JsonNode}) properties there is
*    {@link #createTreeDeserializer(DeserializationConfig, DeserializerProvider, JavaType, BeanProperty)}
* <li>For enumerated types ({@link java.lang.Enum}) there is
*    {@link #createEnumDeserializer(DeserializationConfig, DeserializerProvider, JavaType, BeanProperty)}
*  </li>
* <li>For all other types, {@link #createBeanDeserializer(DeserializationConfig, DeserializerProvider, JavaType,
BeanProperty)}
*   is used.
* </ul>
*<p>
* All above methods take 2 type arguments, except for the first one
* which takes just a single argument.
*/
public abstract class DeserializerFactory
{
  protected final static Deserializers[] NO_DESERIALIZERS = new Deserializers[0];

  /*
  /**********************************************************
  /* Helper class to contain configuration settings
  /**********************************************************
   */

  /**
   * Configuration settings container class for bean deserializer factory
   *
   * @since 1.7
   */
  public abstract static class Config
  {
     /**
      * Fluent/factory method used to construct a configuration object that
      * has same deserializer providers as this instance, plus one specified
      * as argument. Additional provider will be added before existing ones,
      * meaning it has priority over existing definitions.
      */
     public abstract Config withAdditionalDeserializers(Deserializers additional);

     /**
      * Fluent/factory method used to construct a configuration object that
      * has same configuration as this instance plus one additional
```

```
     * deserialiazer modifier. Added modified has highest priority (that is, it
     * gets called before any already registered modifier).
     */
    public abstract Config withDeserializerModifier(BeanDeserializerModifier modifier);

    public abstract Iterable<Deserializers> deserializers();

    public abstract Iterable<BeanDeserializerModifier> deserializerModifiers();

    public abstract boolean hasDeserializers();

    public abstract boolean hasDeserializerModifiers();
}

/*
/**********************************************************
/* Configuration handling
/**********************************************************
 */

/**
 * @since 1.7
 */
public abstract Config getConfig();

/**
 * Method used for creating a new instance of this factory, but with different
 * configuration. Reason for specifying factory method (instead of plain constructor)
 * is to allow proper sub-classing of factories.
 *<p>
 * Note that custom sub-classes <b>must override</b> implementation
 * of this method, as it usually requires instantiating a new instance of
 * factory type. Check out javadocs for
 * {@link org.codehaus.jackson.map.deser.BeanDeserializerFactory} for more details.
 *
 * @since 1.7
 */
public abstract DeserializerFactory withConfig(Config config);

/**
 * Convenience method for creating a new factory instance with additional deserializer
 * provider.
 *
 * @since 1.7
 */
public final DeserializerFactory withAdditionalDeserializers(Deserializers additional) {
    return withConfig(getConfig().withAdditionalDeserializers(additional));
}
```

```
    /**
     * Convenience method for creating a new factory instance with additional
     * {@link BeanDeserializerModifier}.
     *
     * @since 1.7
     */
    public final DeserializerFactory withDeserializerModifier(BeanDeserializerModifier modifier) {
        return withConfig(getConfig().withDeserializerModifier(modifier));
    }

    /*
    /**********************************************************
    /* Basic DeserializerFactory API:
    /**********************************************************
     */

    /**
     * Method called to create (or, for completely immutable deserializers,
     * reuse) a deserializer that can convert JSON content into values of
     * specified Java "bean" (POJO) type.
     * At this point it is known that the type is not otherwise recognized
     * as one of structured types (array, Collection, Map) or a well-known
     * JDK type (enum, primitives/wrappers, String); this method only
     * gets called if other options are exhausted. This also means that
     * this method can be overridden to add support for custom types.
     *
     * @param type Type to be deserialized
     * @param p Provider that can be called to create deserializers for
     *   contained member types
     */
    public abstract JsonDeserializer<Object> createBeanDeserializer(DeserializationConfig config,
DeserializerProvider p,
            JavaType type, BeanProperty property)
        throws JsonMappingException;

    /**
     * Method called to create (or, for completely immutable deserializers,
     * reuse) a deserializer that can convert JSON content into values of
     * specified Java type.
     *
     * @param type Type to be deserialized
     * @param p Provider that can be called to create deserializers for
     *   contained member types
     */
    public abstract JsonDeserializer<?> createArrayDeserializer(DeserializationConfig config, DeserializerProvider p,
            ArrayType type, BeanProperty property)
        throws JsonMappingException;
```

```
  public abstract JsonDeserializer<?> createCollectionDeserializer(DeserializationConfig config,
DeserializerProvider p,
        CollectionType type, BeanProperty property)
    throws JsonMappingException;

  public abstract JsonDeserializer<?> createEnumDeserializer(DeserializationConfig config,DeserializerProvider p,
        JavaType type, BeanProperty property)
    throws JsonMappingException;

  public abstract JsonDeserializer<?> createMapDeserializer(DeserializationConfig config, DeserializerProvider p,
        MapType type, BeanProperty property)
    throws JsonMappingException;

  /**
   * Method called to create and return a deserializer that can construct
   * JsonNode(s) from JSON content.
   */
  public abstract JsonDeserializer<?> createTreeDeserializer(DeserializationConfig config, DeserializerProvider p,
        JavaType type, BeanProperty property)
    throws JsonMappingException;

  /**
   * Method called to find and create a type information deserializer for given base type,
   * if one is needed. If not needed (no polymorphic handling configured for type),
   * should return null.
   *<p>
   * Note that this method is usually only directly called for values of container (Collection,
   * array, Map) types and root values, but not for bean property values.
   *
   * @param baseType Declared base type of the value to deserializer (actual
   *    deserializer type will be this type or its subtype)
   *
   * @return Type deserializer to use for given base type, if one is needed; null if not.
   *
   * @since 1.5
   */
  public TypeDeserializer findTypeDeserializer(DeserializationConfig config, JavaType baseType,
        BeanProperty property)
  {
    // Default implementation returns null for backwards compatibility reasons
    return null;
  }

  /*
  /**********************************************************
  /* Older deprecated versions of creator methods
  /**********************************************************
```

```
     */

    /**
     *<p>
     * Note: declared final to prevent sub-classes from overriding; choice is between
     * hard compile-time error and nastier runtime errors as this method should
     * not be called by core framework any more.
     *
     * @deprecated Since 1.7 should use method that takes in property definition
     */
    @Deprecated
    final
    public TypeDeserializer findTypeDeserializer(DeserializationConfig config, JavaType baseType)
    {
        return findTypeDeserializer(config, baseType, null);
    }

    /**
     *<p>
     * Note: declared final to prevent sub-classes from overriding; choice is between
     * hard compile-time error and nastier runtime errors as this method should
     * not be called by core framework any more.
     * @deprecated Since 1.7 should use method that takes in property definition
     */
    @Deprecated
    final
    public JsonDeserializer<Object> createBeanDeserializer(DeserializationConfig config, JavaType type,
DeserializerProvider p)
        throws JsonMappingException
    {
        return createBeanDeserializer(config, p, type, null);
    }

    /**
     *<p>
     * Note: declared final to prevent sub-classes from overriding; choice is between
     * hard compile-time error and nastier runtime errors as this method should
     * not be called by core framework any more.
     *
     * @deprecated Since 1.7 should use method that takes in property definition
     */
    @Deprecated
    final
    public JsonDeserializer<?> createArrayDeserializer(DeserializationConfig config, ArrayType type,
DeserializerProvider p)
        throws JsonMappingException
    {
        return createArrayDeserializer(config, p, type, null);
```

```
    }

    /**
     *<p>
     * Note: declared final to prevent sub-classes from overriding; choice is between
     * hard compile-time error and nastier runtime errors as this method should
     * not be called by core framework any more.
     *
     * @deprecated Since 1.7 should use method that takes in property definition
     */
    @Deprecated
    final
    public JsonDeserializer<?> createCollectionDeserializer(DeserializationConfig config, CollectionType type,
DeserializerProvider p)
        throws JsonMappingException
    {
        return createCollectionDeserializer(config, p, type, null);
    }

    /**
     *<p>
     * Note: declared final to prevent sub-classes from overriding; choice is between
     * hard compile-time error and nastier runtime errors as this method should
     * not be called by core framework any more.
     *
     * @deprecated Since 1.7 should use method that takes in property definition
     */
    @Deprecated
    final
    public JsonDeserializer<?> createEnumDeserializer(DeserializationConfig config, Class<?> enumClass,
DeserializerProvider p)
        throws JsonMappingException
    {
        return createEnumDeserializer(config, p, TypeFactory.type(enumClass), null);
    }

    /**
     *<p>
     * Note: declared final to prevent sub-classes from overriding; choice is between
     * hard compile-time error and nastier runtime errors as this method should
     * not be called by core framework any more.
     *
     * @deprecated Since 1.7 should use method that takes in property definition
     */
    @Deprecated
    final
    public JsonDeserializer<?> createMapDeserializer(DeserializationConfig config, MapType type,
DeserializerProvider p)
```

```
        throws JsonMappingException
    {
        return createMapDeserializer(config, p, type, null);
    }

    /**
     *<p>
     * Note: declared final to prevent sub-classes from overriding; choice is between
     * hard compile-time error and nastier runtime errors as this method should
     * not be called by core framework any more.
     *
     * @deprecated Since 1.7 should use method that takes in property definition
     */
    @Deprecated
    final
    public JsonDeserializer<?> createTreeDeserializer(DeserializationConfig config, Class<? extends JsonNode>
nodeClass, DeserializerProvider p)
        throws JsonMappingException
    {
        return createTreeDeserializer(config, p, TypeFactory.type(nodeClass), null);
    }
}
package org.codehaus.jackson.map;

import org.codehaus.jackson.map.deser.BeanDeserializerModifier;
import org.codehaus.jackson.type.JavaType;

/**
* Abstract class that defines API used by {@link ObjectMapper} and
* {@link JsonDeserializer}s to obtain deserializers capable of
* re-constructing instances of handled type from JSON content.
*/
public abstract class DeserializerProvider
{
    protected DeserializerProvider() { }

    /**
     * Method that is to configure {@link DeserializerFactory} that provider has
     * to use specified deserializer provider, with highest precedence (that is,
     * additional providers have higher precedence than default one or previously
     * added ones)
     *
     * @since 1.7
     */
    public abstract DeserializerProvider withAdditionalDeserializers(Deserializers d);

    /**
     * @since 1.7
```

```
 */
public abstract DeserializerProvider withDeserializerModifier(BeanDeserializerModifier modifier);

/*
/**********************************************************
/* General deserializer locating method
/**********************************************************
 */

/**
 * Method called to get hold of a deserializer for a value of given type;
 * or if no such deserializer can be found, a default handler (which
 * may do a best-effort generic serialization or just simply
 * throw an exception when invoked).
 *<p>
 * Note: this method is only called for value types; not for keys.
 * Key deserializers can be accessed using {@link #findKeyDeserializer}.
 *
 * @param config Deserialization configuration
 * @param propertyType Declared type of the value to deserializer (obtained using
 *   'setter' method signature and/or type annotations
 * @param property Object that represents accessor for property value; field,
 *    setter method or constructor parameter.
 *
 * @throws JsonMappingException if there are fatal problems with
 *   accessing suitable deserializer; including that of not
 *   finding any serializer
 */
public abstract JsonDeserializer<Object> findValueDeserializer(DeserializationConfig config,
      JavaType propertyType, BeanProperty property)
   throws JsonMappingException;

/**
 * Method called to locate deserializer for given type, as well as matching
 * type deserializer (if one is needed); and if type deserializer is needed,
 * construct a "wrapped" deserializer that can extract and use type information
 * for calling actual deserializer.
 *<p>
 * Since this method is only called for root elements, no referral information
 * is taken.
 *
 * @since 1.5
 */
public abstract JsonDeserializer<Object> findTypedValueDeserializer(DeserializationConfig config,
      JavaType type, BeanProperty property)
   throws JsonMappingException;

/**
```

```
     * Method called to get hold of a deserializer to use for deserializing
     * keys for {@link java.util.Map}.
     *
     * @throws JsonMappingException if there are fatal problems with
     *   accessing suitable key deserializer; including that of not
     *   finding any serializer
     */
    public abstract KeyDeserializer findKeyDeserializer(DeserializationConfig config,
            JavaType type, BeanProperty property)
        throws JsonMappingException;

    /**
     * Method called to find out whether provider would be able to find
     * a deserializer for given type, using a root reference (i.e. not
     * through fields or membership in an array or collection)
     */
    public abstract boolean hasValueDeserializerFor(DeserializationConfig config, JavaType type);

    /*
    /**********************************************************
    /* Deprecated deserializer locating methods (pre-1.7)
    /**********************************************************
     */

    /**
     * Deprecated version of accessor method that was used before version 1.7.
     * Implemented as final to ensure that existing code does not accidentally
     * try to redefine it (given that it is not called by core mapper code)
     *
     * @deprecated As of version 1.7, use version that exposes property object
     *    instead of just its type (needed for contextual deserializers)
     */
    @Deprecated
    public final JsonDeserializer<Object> findValueDeserializer(DeserializationConfig config,
            JavaType type, JavaType referrer, String refPropName)
        throws JsonMappingException
    {
        return findValueDeserializer(config, type, (BeanProperty) null);
    }

    /**
     * Deprecated version of accessor method that was used before version 1.7.
     * Implemented as final to ensure that existing code does not accidentally
     * try to redefine it (given that it is not called by core mapper code)
     *
     * @deprecated As of version 1.7, use version that exposes context class
     *    and property, instead of just types
     *
```

```
 * @since 1.5
 */
@Deprecated
public final JsonDeserializer<Object> findTypedValueDeserializer(DeserializationConfig config,
        JavaType type)
    throws JsonMappingException
{
    return findTypedValueDeserializer(config, type, null);
}

/**
 * Deprecated version of accessor method that was used before version 1.7.
 * Implemented as final to ensure that existing code does not accidentally
 * try to redefine it (given that it is not called by core mapper code)
 *
 * @deprecated As of version 1.7, use version that exposes context class
 *    and property, instead of just types
 *
 * @since 1.5
 */
@Deprecated
public final KeyDeserializer findKeyDeserializer(DeserializationConfig config, JavaType type)
    throws JsonMappingException
{
    return findKeyDeserializer(config, type, null);
}

/*
/**********************************************************
/* Access to caching aspects
/**********************************************************
 */

/**
 * Method that can be used to determine how many deserializers this
 * provider is caching currently
 * (if it does caching: default implementation does)
 * Exact count depends on what kind of deserializers get cached;
 * default implementation caches only dynamically constructed deserializers,
 * but not eagerly constructed standard deserializers (which is different
 * from how serializer provider works).
 *<p>
 * The main use case for this method is to allow conditional flushing of
 * deserializer cache, if certain number of entries is reached.
 *
 * @since 1.4
 */
public abstract int cachedDeserializersCount();
```

```
  /**
   * Method that will drop all dynamically constructed deserializers (ones that
   * are counted as result value for {@link #cachedDeserializersCount}).
   * This can be used to remove memory usage (in case some deserializers are
   * only used once or so), or to force re-construction of deserializers after
   * configuration changes for mapper than owns the provider.
   *
   * @since 1.4
   */
  public abstract void flushCachedDeserializers();
}
package org.codehaus.jackson.map;

import java.lang.annotation.Annotation;
import java.util.*;

import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.JsonSerializer;
import org.codehaus.jackson.map.annotate.JsonSerialize;
import org.codehaus.jackson.map.introspect.*;
import org.codehaus.jackson.map.jsontype.NamedType;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;

/**
* Abstract class that defines API used for introspecting annotation-based
* configuration for serialization and deserialization. Separated
* so that different sets of annotations can be supported, and support
* plugged-in dynamically.
*<p>
* NOTE: due to rapid addition of new methods (and changes to existing methods),
* it is <b>strongly</b> recommended that custom implementations should not directly
* extend this class, but rather extend {@link NopAnnotationIntrospector}.
* This way added methods will not break backwards compatibility of custom annotation
* introspectors.
*/
public abstract class AnnotationIntrospector
{
  /*
  /**********************************************************
  /* Helper types
  /**********************************************************
   */

  /**
   * Value type used with managed and back references; contains type and
   * logic name, used to link related references
```

```
     *
     * @since 1.6
     */
    public static class ReferenceProperty
    {
        public enum Type {
            /**
             * Reference property that Jackson manages and that is serialized normally (by serializing
             * reference object), but is used for resolving back references during
             * deserialization.
             * Usually this can be defined by using
             * {@link org.codehaus.jackson.annotate.JsonManagedReference}
             */
            MANAGED_REFERENCE

            /**
             * Reference property that Jackson manages by suppressing it during serialization,
             * and reconstructing during deserialization.
             * Usually this can be defined by using
             * {@link org.codehaus.jackson.annotate.JsonBackReference}
             */
            ,BACK_REFERENCE
            ;
        }

        private final Type _type;
        private final String _name;

        public ReferenceProperty(Type t, String n) {
            _type = t;
            _name = n;
        }

        public static ReferenceProperty managed(String name) { return new
    ReferenceProperty(Type.MANAGED_REFERENCE, name); }
        public static ReferenceProperty back(String name) { return new
    ReferenceProperty(Type.BACK_REFERENCE, name); }

        public Type getType() { return _type; }
        public String getName() { return _name; }

        public boolean isManagedReference() { return _type == Type.MANAGED_REFERENCE; }
        public boolean isBackReference() { return _type == Type.BACK_REFERENCE; }
    }

    /*
    /**********************************************************
    /* Factory methods
```

```
/**********************************************************
 */

/**
 * Factory method for accessing "no operation" implementation
 * of introspector: instance that will never find any annotation-based
 * configuration.
 *
 * @since 1.3
 */
public static AnnotationIntrospector nopInstance() {
   return NopAnnotationIntrospector.instance;
}

public static AnnotationIntrospector pair(AnnotationIntrospector a1, AnnotationIntrospector a2) {
   return new Pair(a1, a2);
}

/*
/**********************************************************
/* Access to possibly chained introspectors (1.7)
/**********************************************************
 */

/**
 * Method that can be used to collect all "real" introspectors that
 * this introspector contains, if any; or this introspector
 * if it is not a container. Used to get access to all container
 * introspectors in their priority order.
 *<p>
 * Default implementation returns a Singleton list with this introspector
 * as contents.
 * This usually works for sub-classes, except for proxy or delegating "container
 * introspectors" which need to override implementation.
 */
public Collection<AnnotationIntrospector> allIntrospectors() {
   return Collections.singletonList(this);
}

/**
 * Method that can be used to collect all "real" introspectors that
 * this introspector contains, if any; or this introspector
 * if it is not a container. Used to get access to all container
 * introspectors in their priority order.
 *<p>
 * Default implementation adds this introspector in result; this usually
 * works for sub-classes, except for proxy or delegating "container
 * introspectors" which need to override implementation.
```

```
 */
public Collection<AnnotationIntrospector> allIntrospectors(Collection<AnnotationIntrospector> result) {
   result.add(this);
   return result;
}

/*
/**********************************************************
/* Generic annotation properties, lookup
/**********************************************************
*/

/**
 * Method called by framework to determine whether given annotation
 * is handled by this introspector.
 */
public abstract boolean isHandled(Annotation ann);

/*
/**********************************************************
/* General class annotations
/**********************************************************
*/

/**
 * Method that checks whether specified class has annotations
 * that indicate that it is (or is not) cachable. Exact
 * semantics depend on type of class annotated and using
 * class (factory or provider).
 *<p>
 * Currently only used
 * with deserializers, to determine whether provider
 * should cache instances, and if no annotations are found,
 * assumes non-cachable instances.
 *
 * @return True, if class is considered cachable within context,
 *   False if not, and null if introspector does not care either
 *   way.
 */
public abstract Boolean findCachability(AnnotatedClass ac);

/**
 * Method for locating name used as "root name" (for use by
 * some serializers when outputting root-level object -- mostly
 * for XML compatibility purposes) for given class, if one
 * is defined. Returns null if no declaration found; can return
 * explicit empty String, which is usually ignored as well as null.
 *
```

```
     * @since 1.3
     */
    public abstract String findRootName(AnnotatedClass ac);

    /**
     * Method for finding list of properties to ignore for given class
     * (null is returned if not specified).
     * List of property names is applied
     * after other detection mechanisms, to filter out these specific
     * properties from being serialized and deserialized.
     *
     * @since 1.4
     */
    public abstract String[] findPropertiesToIgnore(AnnotatedClass ac);

    /**
     * Method for checking whether an annotation indicates that all unknown properties
     *
     * @since 1.4
     */
    public abstract Boolean findIgnoreUnknownProperties(AnnotatedClass ac);

    /**
     * Method for checking whether properties that have specified type
     * (class, not generics aware) should be completely ignored for
     * serialization and deserialization purposes.
     *
     * @param ac Type to check
     *
     * @return Boolean.TRUE if properties of type should be ignored;
     *   Boolean.FALSE if they are not to be ignored, null for default
     *   handling (which is 'do not ignore')
     *
     * @since 1.7
     */
    public Boolean isIgnorableType(AnnotatedClass ac) {
        return null;
    }

    /**
     * Method for finding if annotated class has associated filter; and if so,
     * to return id that is used to locate filter.
     *
     * @return Id of the filter to use for filtering properties of annotated
     *    class, if any; or null if none found.
     */
    public Object findFilterId(AnnotatedClass ac) {
        return null;
```

```
}

/*
/**********************************************************
/* Property auto-detection
/**********************************************************
 */

/**
 * Method for checking if annotations indicate changes to minimum visibility levels
 * needed for auto-detecting property elements (fields, methods, constructors).
 * A baseline checker is given, and introspector is to either return it as is (if
 * no annotations are found), or build and return a derived instance (using checker's build
 * methods).
 *
 *  @since 1.5
 */
public abstract VisibilityChecker<?> findAutoDetectVisibility(AnnotatedClass ac,
      VisibilityChecker<?> baseChecker);

/*
/**********************************************************
/* Class annotations for PM type handling (1.5+)
/**********************************************************
*/

/**
 * Method for checking if given class has annotations that indicate
 * that specific type resolver is to be used for handling instances.
 * This includes not only
 * instantiating resolver builder, but also configuring it based on
 * relevant annotations (not including ones checked with a call to
 * {@link #findSubtypes}
 *
 * @param ac Annotated class to check for annotations
 * @param baseType Base java type of value for which resolver is to be found
 *
 * @return Type resolver builder for given type, if one found; null if none
 *
 * @since 1.5
 */
public TypeResolverBuilder<?> findTypeResolver(AnnotatedClass ac, JavaType baseType) {
   return null;
}

/**
 * Method for checking if given property entity (field or method) has annotations
 * that indicate that specific type resolver is to be used for handling instances.
```

```
     * This includes not only
     * instantiating resolver builder, but also configuring it based on
     * relevant annotations (not including ones checked with a call to
     * {@link #findSubtypes}
     *
     * @param am Annotated member (field or method) to check for annotations
     * @param baseType Base java type of property for which resolver is to be found
     *
     * @return Type resolver builder for properties of given entity, if one found;
     *    null if none
     *
     * @since 1.5
     */
    public TypeResolverBuilder<?> findPropertyTypeResolver(AnnotatedMember am, JavaType baseType) {
        return null;
    }

    /**
     * Method for checking if given structured property entity (field or method that
     * has nominal value of Map, Collection or array type) has annotations
     * that indicate that specific type resolver is to be used for handling type
     * information of contained values.
     * This includes not only
     * instantiating resolver builder, but also configuring it based on
     * relevant annotations (not including ones checked with a call to
     * {@link #findSubtypes}
     *
     * @param am Annotated member (field or method) to check for annotations
     * @param containerType Type of property for which resolver is to be found (must be a container type)
     *
     * @return Type resolver builder for values contained in properties of given entity,
     *    if one found; null if none
     *
     * @since 1.5
     */
    public TypeResolverBuilder<?> findPropertyContentTypeResolver(AnnotatedMember am, JavaType
containerType) {
        return null;
    }

    /**
     * Method for locating annotation-specified subtypes related to annotated
     * entity (class, method, field). Note that this is only guaranteed to be
     * a list of directly
     * declared subtypes, no recursive processing is guarantees (i.e. caller
     * has to do it if/as necessary)
     *
     * @param a Annotated entity (class, field/method) to check for annotations
```

```
     *
     * @since 1.5
     */
    public List<NamedType> findSubtypes(Annotated a) {
        return null;
    }

    /**
     * Method for checking if specified type has explicit name.
     *
     * @param ac Class to check for type name annotations
     *
     * @since 1.5
     */
    public String findTypeName(AnnotatedClass ac) {
        return null;
    }

    /*
    /**********************************************************
    /* General member (field, method/constructor) annotations
    /**********************************************************
     */

    /**
     * Note: defined as non-abstract to reduce fragility between
     * versions.
     *
     * @since 1.6
     */
    public ReferenceProperty findReferenceType(AnnotatedMember member) {
        return null;
    }

    /*
    /**********************************************************
    /* General method annotations
    /**********************************************************
     */

    /**
     * Method for checking whether there is an annotation that
     * indicates that given method should be ignored for all
     * operations (serialization, deserialization).
     *<p>
     * Note that this method should <b>ONLY</b> return true for such
     * explicit ignoral cases; and not if method just happens not to
     * be visible for annotation processor.
```

```
     *
     * @return True, if an annotation is found to indicate that the
     *    method should be ignored; false if not.
     */
    public abstract boolean isIgnorableMethod(AnnotatedMethod m);

    /**
     * @since 1.2
     */
    public abstract boolean isIgnorableConstructor(AnnotatedConstructor c);

    /*
    /**********************************************************
    /* General field annotations
    /**********************************************************
     */

    /**
     * Method for checking whether there is an annotation that
     * indicates that given field should be ignored for all
     * operations (serialization, deserialization).
     *
     * @return True, if an annotation is found to indicate that the
     *    field should be ignored; false if not.
     */
    public abstract boolean isIgnorableField(AnnotatedField f);

    /*
    /**********************************************************
    /* Serialization: general annotations
    /**********************************************************
     */

    /**
     * Method for getting a serializer definition on specified method
     * or field. Type of definition is either instance (of type
     * {@link JsonSerializer}) or Class (of type
     * <code>Class<JsonSerializer></code>); if value of different
     * type is returned, a runtime exception may be thrown by caller.
     *
     * @deprecated Should use version that gets property object
     */
    @Deprecated
    public Object findSerializer(Annotated am) {
       return findSerializer(am, null);
    }

    /**
```

```
     * Method for getting a serializer definition on specified method
     * or field. Type of definition is either instance (of type
     * {@link JsonSerializer}) or Class (of type
     * <code>Class<JsonSerializer></code>); if value of different
     * type is returned, a runtime exception may be thrown by caller.
     */
    public abstract Object findSerializer(Annotated am, BeanProperty property);

    /**
     * Method for checking whether given annotated entity (class, method,
     * field) defines which Bean/Map properties are to be included in
     * serialization.
     * If no annotation is found, method should return given second
     * argument; otherwise value indicated by the annotation
     *
     * @return Enumerated value indicating which properties to include
     *   in serialization
     */
    public abstract JsonSerialize.Inclusion findSerializationInclusion(Annotated a, JsonSerialize.Inclusion defValue);

    /**
     * Method for accessing annotated type definition that a
     * method/field can have, to be used as the type for serialization
     * instead of the runtime type.
     * Type returned (if any) needs to be widening conversion (super-type).
     * Declared return type of the method is also considered acceptable.
     *
     * @return Class to use instead of runtime type
     */
    public abstract Class<?> findSerializationType(Annotated a);

    /**
     * Method for accessing declared typing mode annotated (if any).
     * This is used for type detection, unless more granular settings
     * (such as actual exact type; or serializer to use which means
     * no type information is needed) take precedence.
     *
     * @since 1.2
     *
     * @return Typing mode to use, if annotation is found; null otherwise
     */
    public abstract JsonSerialize.Typing findSerializationTyping(Annotated a);

    /**
     * Method for checking if annotated serializable property (represented by
     * field or getter method) has definitions for views it is to be included
     * in. If null is returned, no view definitions exist and property is always
     * included; otherwise it will only be included for views included in returned
```

```
 * array. View matches are checked using class inheritance rules (sub-classes
 * inherit inclusions of super-classes)
 *
 * @param a Annotated serializable property (field or getter method)
 * @return Array of views (represented by classes) that the property is included in;
 *    if null, always included (same as returning array containing <code>Object.class</code>)
 */
public abstract Class<?>[] findSerializationViews(Annotated a);

/*
/**********************************************************
/* Serialization: class annotations
/**********************************************************
 */

/**
 * Method for accessing defined property serialization order (which may be
 * partial). May return null if no ordering is defined.
 *
 * @since 1.4
 */
public abstract String[] findSerializationPropertyOrder(AnnotatedClass ac);

/**
 * Method for checking whether an annotation indicates that serialized properties
 * for which no explicit is defined should be alphabetically (lexicograpically)
 * ordered
 *
 * @since 1.4
 */
public abstract Boolean findSerializationSortAlphabetically(AnnotatedClass ac);

/*
/**********************************************************
/* Serialization: method annotations
/**********************************************************
 */

/**
 * Method for checking whether given method has an annotation
 * that suggests property name associated with method that
 * may be a "getter". Should return null if no annotation
 * is found; otherwise a non-null String.
 * If non-null value is returned, it is used as the property
 * name, except for empty String ("") which is taken to mean
 * "use standard bean name detection if applicable;
 * method name if not".
 */
```

```
public abstract String findGettablePropertyName(AnnotatedMethod am);

/**
 * Method for checking whether given method has an annotation
 * that suggests that the return value of annotated method
 * should be used as "the value" of the object instance; usually
 * serialized as a primitive value such as String or number.
 *
 * @return True if such annotation is found (and is not disabled);
 *   false if no enabled annotation is found
 */
public abstract boolean hasAsValueAnnotation(AnnotatedMethod am);

/**
 * Method for determining the String value to use for serializing
 * given enumeration entry; used when serializing enumerations
 * as Strings (the standard method).
 *
 * @return Serialized enum value.
 */
public abstract String findEnumValue(Enum<?> value);

/*
/**********************************************************
/* Serialization: field annotations
/**********************************************************
 */

/**
 * Method for checking whether given member field represent
 * a serializable logical property; and if so, returns the
 * name of that property.
 * Should return null if no annotation is found (indicating it
 * is not a serializable field); otherwise a non-null String.
 * If non-null value is returned, it is used as the property
 * name, except for empty String ("") which is taken to mean
 * "use the field name as is".
 */
public abstract String findSerializablePropertyName(AnnotatedField af);

/*
/**********************************************************
/* Deserialization: general annotations
/**********************************************************
 */

/**
 * @deprecated Used version that takes property
```

```
 */
@Deprecated
public final Object findDeserializer(Annotated am) {
   return findDeserializer(am, null);
}

/**
 * Method for getting a deserializer definition on specified method
 * or field.
 * Type of definition is either instance (of type
 * {@link JsonDeserializer}) or Class (of type
 * <code>Class<JsonDeserializer></code>); if value of different
 * type is returned, a runtime exception may be thrown by caller.
 */
public abstract Object findDeserializer(Annotated am, BeanProperty property);

/**
 * Method for getting a deserializer definition for keys of
 * associated <code>Map</code> property.
 * Type of definition is either instance (of type
 * {@link JsonDeserializer}) or Class (of type
 * <code>Class<JsonDeserializer></code>); if value of different
 * type is returned, a runtime exception may be thrown by caller.
 *
 * @since 1.3
 */
public abstract Class<? extends KeyDeserializer> findKeyDeserializer(Annotated am);

/**
 * Method for getting a deserializer definition for content (values) of
 * associated <code>Collection</code>, <code>array</code> or
 * <code>Map</code> property.
 * Type of definition is either instance (of type
 * {@link JsonDeserializer}) or Class (of type
 * <code>Class<JsonDeserializer></code>); if value of different
 * type is returned, a runtime exception may be thrown by caller.
 *
 * @since 1.3
 */
public abstract Class<? extends JsonDeserializer<?>> findContentDeserializer(Annotated am);

/**
 * Method for accessing annotated type definition that a
 * method can have, to be used as the type for serialization
 * instead of the runtime type.
 * Type must be a narrowing conversion
 * (i.e.subtype of declared type).
 * Declared return type of the method is also considered acceptable.
```

```
     *
     * @param baseType Assumed type before considering annotations
     * @param propName Logical property name of the property that uses
     *    type, if known; null for types not associated with property
     *
     * @return Class to use for deserialization instead of declared type
     */
    public abstract Class<?> findDeserializationType(Annotated am, JavaType baseType,
            String propName);

    /**
     * Method for accessing additional narrowing type definition that a
     * method can have, to define more specific key type to use.
     * It should be only be used with {@link java.util.Map} types.
     *
     * @param baseKeyType Assumed key type before considering annotations
     * @param propName Logical property name of the property that uses
     *    type, if known; null for types not associated with property
     *
     * @return Class specifying more specific type to use instead of
     *   declared type, if annotation found; null if not
     */
    public abstract Class<?> findDeserializationKeyType(Annotated am, JavaType baseKeyType,
            String propName);

    /**
     * Method for accessing additional narrowing type definition that a
     * method can have, to define more specific content type to use;
     * content refers to Map values and Collection/array elements.
     * It should be only be used with Map, Collection and array types.
     *
     * @param baseContentType Assumed content (value) type before considering annotations
     * @param propName Logical property name of the property that uses
     *    type, if known; null for types not associated with property
     *
     * @return Class specifying more specific type to use instead of
     *   declared type, if annotation found; null if not
     */
    public abstract Class<?> findDeserializationContentType(Annotated am, JavaType baseContentType,
            String propName);

    /*
    /**********************************************************
    /* Deserialization: class annotations
    /**********************************************************
    */

    /*
```

```
/**********************************************************
/* Deserialization: method annotations
/**********************************************************
 */

/**
 * Method for checking whether given method has an annotation
 * that suggests property name associated with method that
 * may be a "setter". Should return null if no annotation
 * is found; otherwise a non-null String.
 * If non-null value is returned, it is used as the property
 * name, except for empty String ("") which is taken to mean
 * "use standard bean name detection if applicable;
 * method name if not".
 */
public abstract String findSettablePropertyName(AnnotatedMethod am);

/**
 * Method for checking whether given method has an annotation
 * that suggests that the method is to serve as "any setter";
 * method to be used for setting values of any properties for
 * which no dedicated setter method is found.
 *
 * @return True if such annotation is found (and is not disabled),
 *   false otherwise
 */
public boolean hasAnySetterAnnotation(AnnotatedMethod am) {
   return false;
}

/**
 * Method for checking whether given method has an annotation
 * that suggests that the method is to serve as "any setter";
 * method to be used for accessing set of miscellaneous "extra"
 * properties, often bound with matching "any setter" method.
 *
 * @return True if such annotation is found (and is not disabled),
 *   false otherwise
 *
 * @since 1.6
 */
public boolean hasAnyGetterAnnotation(AnnotatedMethod am) {
   return false;
}

/**
 * Method for checking whether given annotated item (method, constructor)
 * has an annotation
```

```
     * that suggests that the method is a "creator" (aka factory)
     * method to be used for construct new instances of deserialized
     * values.
     *
     * @return True if such annotation is found (and is not disabled),
     *   false otherwise
     */
    public boolean hasCreatorAnnotation(Annotated a) {
       return false;
    }

    /*
    /**********************************************************
    /* Deserialization: field annotations
    /**********************************************************
     */

    /**
     * Method for checking whether given member field represent
     * a deserializable logical property; and if so, returns the
     * name of that property.
     * Should return null if no annotation is found (indicating it
     * is not a deserializable field); otherwise a non-null String.
     * If non-null value is returned, it is used as the property
     * name, except for empty String ("") which is taken to mean
     * "use the field name as is".
     */
    public abstract String findDeserializablePropertyName(AnnotatedField af);

    /*
    /**********************************************************
    /* Deserialization: parameter annotations (for
    /* creator method parameters)
    /**********************************************************
     */

    /**
     * Method for checking whether given set of annotations indicates
     * property name for associated parameter.
     * No actual parameter object can be passed since JDK offers no
     * representation; just annotations.
     */
    public abstract String findPropertyNameForParam(AnnotatedParameter param);

    /*
    /**********************************************************
    /* Helper classes
    /**********************************************************
```

```
 */

/**
 * Helper class that allows using 2 introspectors such that one
 * introspector acts as the primary one to use; and second one
 * as a fallback used if the primary does not provide conclusive
 * or useful result for a method.
 *<p>
 * An obvious consequence of priority is that it is easy to construct
 * longer chains of introspectors by linking multiple pairs.
 * Currently most likely combination is that of using the default
 * Jackson provider, along with JAXB annotation introspector (available
 * since version 1.1).
 */
public static class Pair
    extends AnnotationIntrospector
{
    protected final AnnotationIntrospector _primary, _secondary;

    public Pair(AnnotationIntrospector p,
                AnnotationIntrospector s)
    {
        _primary = p;
        _secondary = s;
    }

    /**
     * Helper method for constructing a Pair from two given introspectors (if
     * neither is null); or returning non-null introspector if one is null
     * (and return just null if both are null)
     *
     * @since 1.7
     */
    public static AnnotationIntrospector create(AnnotationIntrospector primary,
            AnnotationIntrospector secondary)
    {
        if (primary == null) {
            return secondary;
        }
        if (secondary == null) {
            return primary;
        }
        return new Pair(primary, secondary);
    }

    @Override
    public Collection<AnnotationIntrospector> allIntrospectors() {
        return allIntrospectors(new ArrayList<AnnotationIntrospector>());
```

```
}

@Override
public Collection<AnnotationIntrospector> allIntrospectors(Collection<AnnotationIntrospector> result)
{
   _primary.allIntrospectors(result);
   _secondary.allIntrospectors(result);
   return result;
}

// // // Generic annotation properties, lookup

@Override
public boolean isHandled(Annotation ann)
{
   return _primary.isHandled(ann) || _secondary.isHandled(ann);
}

/*
/**********************************************************
/* General class annotations
/**********************************************************
 */

@Override
public Boolean findCachability(AnnotatedClass ac)
{
   Boolean result = _primary.findCachability(ac);
   if (result == null) {
      result = _secondary.findCachability(ac);
   }
   return result;
}

@Override
public String findRootName(AnnotatedClass ac)
{
   String name1 = _primary.findRootName(ac);
   if (name1 == null) {
      return _secondary.findRootName(ac);
   } else if (name1.length() > 0) {
      return name1;
   }
   // name1 is empty; how about secondary?
   String name2 = _secondary.findRootName(ac);
   return (name2 == null) ? name1 : name2;
}
```

```
@Override
public String[] findPropertiesToIgnore(AnnotatedClass ac)
{
   String[] result = _primary.findPropertiesToIgnore(ac);
   if (result == null) {
      result = _secondary.findPropertiesToIgnore(ac);
   }
   return result;
}

@Override
public Boolean findIgnoreUnknownProperties(AnnotatedClass ac)
{
   Boolean result = _primary.findIgnoreUnknownProperties(ac);
   if (result == null) {
      result = _secondary.findIgnoreUnknownProperties(ac);
   }
   return result;
}

@Override
public Boolean isIgnorableType(AnnotatedClass ac)
{
   Boolean result = _primary.isIgnorableType(ac);
   if (result == null) {
      result = _secondary.isIgnorableType(ac);
   }
   return result;
}

@Override
public Object findFilterId(AnnotatedClass ac)
{
   Object id = _primary.findFilterId(ac);
   if (id == null) {
      id = _secondary.findFilterId(ac);
   }
   return id;
}

/*
/**********************************************************
/* Property auto-detection
/**********************************************************
*/

@Override
public VisibilityChecker<?> findAutoDetectVisibility(AnnotatedClass ac,
```

```
        VisibilityChecker<?> checker)
    {
        /* Note: to have proper priorities, we must actually call delegatees
         * in reverse order:
         */
        checker = _secondary.findAutoDetectVisibility(ac, checker);
        return _primary.findAutoDetectVisibility(ac, checker);
    }

    /*
    /**********************************************************
    /* Type handling
    /**********************************************************
    */

    @Override
    public TypeResolverBuilder<?> findTypeResolver(AnnotatedClass ac, JavaType baseType)
    {
        TypeResolverBuilder<?> b = _primary.findTypeResolver(ac, baseType);
        if (b == null) {
            b = _secondary.findTypeResolver(ac, baseType);
        }
        return b;
    }

    @Override
    public TypeResolverBuilder<?> findPropertyTypeResolver(AnnotatedMember am, JavaType baseType)
    {
        TypeResolverBuilder<?> b = _primary.findPropertyTypeResolver(am, baseType);
        if (b == null) {
            b = _secondary.findPropertyTypeResolver(am, baseType);
        }
        return b;
    }

    @Override
    public TypeResolverBuilder<?> findPropertyContentTypeResolver(AnnotatedMember am, JavaType
baseType)
    {
        TypeResolverBuilder<?> b = _primary.findPropertyContentTypeResolver(am, baseType);
        if (b == null) {
            b = _secondary.findPropertyContentTypeResolver(am, baseType);
        }
        return b;
    }

    @Override
    public List<NamedType> findSubtypes(Annotated a)
```

```
{
   List<NamedType> types1 = _primary.findSubtypes(a);
   List<NamedType> types2 = _secondary.findSubtypes(a);
   if (types1 == null || types1.isEmpty()) return types2;
   if (types2 == null || types2.isEmpty()) return types1;
   ArrayList<NamedType> result = new ArrayList<NamedType>(types1.size() + types2.size());
   result.addAll(types1);
   result.addAll(types2);
   return result;
}

@Override
public String findTypeName(AnnotatedClass ac)
{
   String name = _primary.findTypeName(ac);
   if (name == null || name.length() == 0) {
      name = _secondary.findTypeName(ac);
   }
   return name;
}

// // // General member (field, method/constructor) annotations

@Override
public ReferenceProperty findReferenceType(AnnotatedMember member)
{
   ReferenceProperty ref = _primary.findReferenceType(member);
   if (ref == null) {
      ref = _secondary.findReferenceType(member);
   }
   return ref;
}

// // // General method annotations

@Override
public boolean isIgnorableMethod(AnnotatedMethod m) {
   return _primary.isIgnorableMethod(m) || _secondary.isIgnorableMethod(m);
}

@Override
public boolean isIgnorableConstructor(AnnotatedConstructor c) {
   return _primary.isIgnorableConstructor(c) || _secondary.isIgnorableConstructor(c);
}

// // // General field annotations

@Override
```

```
public boolean isIgnorableField(AnnotatedField f)
{
   return _primary.isIgnorableField(f) || _secondary.isIgnorableField(f);
}

// // // Serialization: general annotations

@Override
public Object findSerializer(Annotated am, BeanProperty property)
{
   Object result = _primary.findSerializer(am, property);
   /* Are there non-null results that should be ignored?
    * (i.e. should some validation be done here)
    * For now let's assume no
    */
   if (result == null) {
      result = _secondary.findSerializer(am, property);
   }
   return result;
}

@Override
public JsonSerialize.Inclusion findSerializationInclusion(Annotated a,
                                          JsonSerialize.Inclusion defValue)
{
   /* This is bit trickier: need to combine results in a meaningful
    * way. Seems like it should be a disjoint; that is, most
    * restrictive value should be returned.
    * For enumerations, comparison is done by indexes, which
    * works: largest value is the last one, which is the most
    * restrictive value as well.
    */
   /* 09-Mar-2010, tatu: Actually, as per [JACKSON-256], it is probably better to just
    *    use strict overriding. Simpler, easier to understand.
    */
   // note: call secondary first, to give lower priority
   defValue = _secondary.findSerializationInclusion(a, defValue);
   defValue = _primary.findSerializationInclusion(a, defValue);
   return defValue;
}

@Override
public Class<?> findSerializationType(Annotated a)
{
   Class<?> result = _primary.findSerializationType(a);
   if (result == null) {
      result = _secondary.findSerializationType(a);
   }
```

```
    return result;
}

@Override
public JsonSerialize.Typing findSerializationTyping(Annotated a)
{
    JsonSerialize.Typing result = _primary.findSerializationTyping(a);
    if (result == null) {
        result = _secondary.findSerializationTyping(a);
    }
    return result;
}

@Override
public Class<?>[] findSerializationViews(Annotated a)
{
    /* Theoretically this could be trickier, if multiple introspectors
     * return non-null entries. For now, though, we'll just consider
     * first one to return non-null to win.
     */
    Class<?>[] result = _primary.findSerializationViews(a);
    if (result == null) {
        result = _secondary.findSerializationViews(a);
    }
    return result;
}

// // // Serialization: class annotations

@Override
public String[] findSerializationPropertyOrder(AnnotatedClass ac) {
    String[] result = _primary.findSerializationPropertyOrder(ac);
    if (result == null) {
        result = _secondary.findSerializationPropertyOrder(ac);
    }
    return result;
}

/**
 * Method for checking whether an annotation indicates that serialized properties
 * for which no explicit is defined should be alphabetically (lexicograpically)
 * ordered
 */
@Override
public Boolean findSerializationSortAlphabetically(AnnotatedClass ac) {
    Boolean result = _primary.findSerializationSortAlphabetically(ac);
    if (result == null) {
        result = _secondary.findSerializationSortAlphabetically(ac);
```

```
    }
    return result;
}

// // // Serialization: method annotations

@Override
public String findGettablePropertyName(AnnotatedMethod am)
{
    String result = _primary.findGettablePropertyName(am);
    if (result == null) {
        result = _secondary.findGettablePropertyName(am);
    } else if (result.length() == 0) {
        /* Empty String is a default; can be overridden by
         * more explicit answer from secondary entry
         */
        String str2 = _secondary.findGettablePropertyName(am);
        if (str2 != null) {
            result = str2;
        }
    }
    return result;
}

@Override
public boolean hasAsValueAnnotation(AnnotatedMethod am)
{
    return _primary.hasAsValueAnnotation(am) || _secondary.hasAsValueAnnotation(am);
}

@Override
public String findEnumValue(Enum<?> value)
{
    String result = _primary.findEnumValue(value);
    if (result == null) {
        result = _secondary.findEnumValue(value);
    }
    return result;
}

// // // Serialization: field annotations

@Override
public String findSerializablePropertyName(AnnotatedField af)
{
    String result = _primary.findSerializablePropertyName(af);
    if (result == null) {
        result = _secondary.findSerializablePropertyName(af);
```

```
    } else if (result.length() == 0) {
        /* Empty String is a default; can be overridden by
         * more explicit answer from secondary entry
         */
        String str2 = _secondary.findSerializablePropertyName(af);
        if (str2 != null) {
            result = str2;
        }
    }
    return result;
}

// // // Deserialization: general annotations

@Override
public Object findDeserializer(Annotated am, BeanProperty property)
{
    Object result = _primary.findDeserializer(am, property);
    if (result == null) {
        result = _secondary.findDeserializer(am, property);
    }
    return result;
}

@Override
public Class<? extends KeyDeserializer> findKeyDeserializer(Annotated am)
{
    Class<? extends KeyDeserializer> result = _primary.findKeyDeserializer(am);
    if (result == null || result == KeyDeserializer.None.class) {
        result = _secondary.findKeyDeserializer(am);
    }
    return result;
}

@Override
public Class<? extends JsonDeserializer<?>> findContentDeserializer(Annotated am)
{
    Class<? extends JsonDeserializer<?>> result = _primary.findContentDeserializer(am);
    if (result == null || result == JsonDeserializer.None.class) {
        result = _secondary.findContentDeserializer(am);
    }
    return result;
}

@Override
public Class<?> findDeserializationType(Annotated am, JavaType baseType,
        String propName)
{
```

```
    Class<?> result = _primary.findDeserializationType(am, baseType, propName);
    if (result == null) {
        result = _secondary.findDeserializationType(am, baseType, propName);
    }
    return result;
}

@Override
public Class<?> findDeserializationKeyType(Annotated am, JavaType baseKeyType,
        String propName)
{
    Class<?> result = _primary.findDeserializationKeyType(am, baseKeyType, propName);
    if (result == null) {
        result = _secondary.findDeserializationKeyType(am, baseKeyType, propName);
    }
    return result;
}

@Override
public Class<?> findDeserializationContentType(Annotated am, JavaType baseContentType,
        String propName)
{
    Class<?> result = _primary.findDeserializationContentType(am, baseContentType, propName);
    if (result == null) {
        result = _secondary.findDeserializationContentType(am, baseContentType, propName);
    }
    return result;
}


// // // Deserialization: method annotations

@Override
public String findSettablePropertyName(AnnotatedMethod am)
{
    String result = _primary.findSettablePropertyName(am);
    if (result == null) {
        result = _secondary.findSettablePropertyName(am);
    } else if (result.length() == 0) {
        /* Empty String is a default; can be overridden by
         * more explicit answer from secondary entry
         */
        String str2 = _secondary.findSettablePropertyName(am);
        if (str2 != null) {
            result = str2;
        }
    }
    return result;
```

```
}

@Override
public boolean hasAnySetterAnnotation(AnnotatedMethod am)
{
   return _primary.hasAnySetterAnnotation(am) || _secondary.hasAnySetterAnnotation(am);
}

@Override
public boolean hasAnyGetterAnnotation(AnnotatedMethod am)
{
   return _primary.hasAnyGetterAnnotation(am) || _secondary.hasAnyGetterAnnotation(am);
}

@Override
public boolean hasCreatorAnnotation(Annotated a)
{
   return _primary.hasCreatorAnnotation(a) || _secondary.hasCreatorAnnotation(a);
}

// // // Deserialization: field annotations

@Override
public String findDeserializablePropertyName(AnnotatedField af)
{
   String result = _primary.findDeserializablePropertyName(af);
   if (result == null) {
      result = _secondary.findDeserializablePropertyName(af);
   } else if (result.length() == 0) {
      /* Empty String is a default; can be overridden by
       * more explicit answer from secondary entry
       */
      String str2 = _secondary.findDeserializablePropertyName(af);
      if (str2 != null) {
         result = str2;
      }
   }
   return result;
}

// // // Deserialization: parameter annotations (for creators)

@Override
public String findPropertyNameForParam(AnnotatedParameter param)
{
   String result = _primary.findPropertyNameForParam(param);
   if (result == null) {
      result = _secondary.findPropertyNameForParam(param);
```

```
        }
        return result;
      }
    }

}
package org.codehaus.jackson.map;

import java.text.DateFormat;
import java.util.*;

import org.codehaus.jackson.Base64Variant;
import org.codehaus.jackson.Base64Variants;
import org.codehaus.jackson.annotate.*;
import org.codehaus.jackson.map.introspect.AnnotatedClass;
import org.codehaus.jackson.map.introspect.NopAnnotationIntrospector;
import org.codehaus.jackson.map.introspect.VisibilityChecker;
import org.codehaus.jackson.map.jsontype.SubtypeResolver;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.map.jsontype.impl.StdSubtypeResolver;
import org.codehaus.jackson.map.type.ClassKey;
import org.codehaus.jackson.map.util.LinkedNode;
import org.codehaus.jackson.map.util.StdDateFormat;
import org.codehaus.jackson.node.JsonNodeFactory;
import org.codehaus.jackson.type.JavaType;

/**
* Object that contains baseline configuration for deserialization
* process. An instance is owned by {@link ObjectMapper}, which makes
* a copy that is passed during serialization process to
* {@link DeserializerProvider} and {@link DeserializerFactory}.
*<p>
* Note: although configuration settings can be changed at any time
* (for factories and instances), they are not guaranteed to have
* effect if called after constructing relevant mapper or deserializer
* instance. This because some objects may be configured, constructed and
* cached first time they are needed.
*/
public class DeserializationConfig
  implements MapperConfig<DeserializationConfig>
{
  /**
   * Enumeration that defines togglable features that guide
   * the serialization feature.
   */
  public enum Feature {
    /*
    /*********************************************************
```

```
     *  Introspection features
    /**********************************************************
     */

    /**
     * Feature that determines whether annotation introspection
     * is used for configuration; if enabled, configured
     * {@link AnnotationIntrospector} will be used: if disabled,
     * no annotations are considered.
     *<P>
     * Feature is enabled by default.
     *
     * @since 1.2
     */
    USE_ANNOTATIONS(true),

    /**
     * Feature that determines whether "setter" methods are
     * automatically detected based on standard Bean naming convention
     * or not. If yes, then all public one-argument methods that
     * start with prefix "set"
     * are considered setters. If disabled, only methods explicitly
     * annotated are considered setters.
     *<p>
     * Note that this feature has lower precedence than per-class
     * annotations, and is only used if there isn't more granular
     * configuration available.
     *<P>
     * Feature is enabled by default.
     */
    AUTO_DETECT_SETTERS(true),

    /**
     * Feature that determines whether "creator" methods are
     * automatically detected by consider public constructors,
     * and static single argument methods with name "valueOf".
     * If disabled, only methods explicitly annotated are considered
     * creator methods (except for the no-arg default constructor which
     * is always considered a factory method).
     *<p>
     * Note that this feature has lower precedence than per-class
     * annotations, and is only used if there isn't more granular
     * configuration available.
     *<P>
     * Feature is enabled by default.
     */
    AUTO_DETECT_CREATORS(true),
```

```
/**
 * Feature that determines whether non-static fields are recognized as
 * properties.
 * If yes, then all public member fields
 * are considered as properties. If disabled, only fields explicitly
 * annotated are considered property fields.
 *<p>
 * Note that this feature has lower precedence than per-class
 * annotations, and is only used if there isn't more granular
 * configuration available.
 *<P>
 * Feature is enabled by default.
 *
 * @since 1.1
 */
AUTO_DETECT_FIELDS(true),

/**
 * Feature that determines whether otherwise regular "getter"
 * methods (but only ones that handle Collections and Maps,
 * not getters of other type)
 * can be used for purpose of getting a reference to a Collection
 * and Map to modify the property, without requiring a setter
 * method.
 * This is similar to how JAXB framework sets Collections and
 * Maps: no setter is involved, just setter.
 *<p>
 * Note that such getters-as-setters methods have lower
 * precedence than setters, so they are only used if no
 * setter is found for the Map/Collection property.
 *<p>
 * Feature is enabled by default.
 */
USE_GETTERS_AS_SETTERS(true),

/**
 * Feature that determines whether method and field access
 * modifier settings can be overridden when accessing
 * properties. If enabled, method
 * {@link java.lang.reflect.AccessibleObject#setAccessible}
 * may be called to enable access to otherwise unaccessible
 * objects.
 */
CAN_OVERRIDE_ACCESS_MODIFIERS(true),

/*
/**********************************************************
 *  Type conversion features
```

```
/*********************************************************
 */

/**
 * Feature that determines whether Json floating point numbers
 * are to be deserialized into {@link java.math.BigDecimal}s
 * if only generic type description (either {@link Object} or
 * {@link Number}, or within untyped {@link java.util.Map}
 * or {@link java.util.Collection} context) is available.
 * If enabled such values will be deserialized as {@link java.math.BigDecimal}s;
 * if disabled, will be deserialized as {@link Double}s.
 * <p>
 * Feature is disabled by default, meaning that "untyped" floating
 * point numbers will by default be deserialized as {@link Double}s
 * (choice is for performance reason -- BigDecimals are slower than
 * Doubles)
 */
USE_BIG_DECIMAL_FOR_FLOATS(false),

/**
 * Feature that determines whether Json integral (non-floating-point)
 * numbers are to be deserialized into {@link java.math.BigInteger}s
 * if only generic type description (either {@link Object} or
 * {@link Number}, or within untyped {@link java.util.Map}
 * or {@link java.util.Collection} context) is available.
 * If enabled such values will be deserialized as
 * {@link java.math.BigInteger}s;
 * if disabled, will be deserialized as "smallest" available type,
 * which is either {@link Integer}, {@link Long} or
 * {@link java.math.BigInteger}, depending on number of digits.
 * <p>
 * Feature is disabled by default, meaning that "untyped" floating
 * point numbers will by default be deserialized using whatever
 * is the most compact integral type, to optimize efficiency.
 */
USE_BIG_INTEGER_FOR_INTS(false),

/**
 * Feature that determines standard deserialization mechanism used for
 * Enum values: if enabled, Enums are assumed to have been serialized  using
 * return value of <code>Enum.toString()</code>;
 * if disabled, return value of <code>Enum.name()</code> is assumed to have been used.
 * Since pre-1.6 method was to use Enum name, this is the default.
 *<p>
 * Note: this feature should usually have same value
 * as {@link SerializationConfig.Feature#WRITE_ENUMS_USING_TO_STRING}.
 *<p>
 * For further details, check out [JACKSON-212]
```

```
     *
     * @since 1.6
     */
    READ_ENUMS_USING_TO_STRING(false),

    /*
    /**********************************************************
     *  Error handling features
    /**********************************************************
     */

    /**
     * Feature that determines whether encountering of unknown
     * properties (ones that do not map to a property, and there is
     * no "any setter" or handler that can handle it)
     * should result in a failure (by throwing a
     * {@link JsonMappingException}) or not.
     * This setting only takes effect after all other handling
     * methods for unknown properties have been tried, and
     * property remains unhandled.
     *<p>
     * Feature is enabled by default, meaning that
     * {@link JsonMappingException} is thrown if an unknown property
     * is encountered. This is the implicit default prior to
     * introduction of the feature.
     *
     * @since 1.2
     */
    FAIL_ON_UNKNOWN_PROPERTIES(true),

    /**
     * Feature that determines whether encountering of JSON null
     * is an error when deserializing into Java primitive types
     * (like 'int' or 'double'). If it is, a JsonProcessingException
     * is thrown to indicate this; if not, default value is used
     * (0 for 'int', 0.0 for double, same defaulting as what JVM uses).
     *<p>
     * Feature is disabled by default (to be consistent with behavior
     * of Jackson 1.6),
     * i.e. to allow use of nulls for primitive properties.
     *
     * @since 1.7
     */
    FAIL_ON_NULL_FOR_PRIMITIVES(false),

    /**
     * Feature that determines whether JSON integer numbers are valid
     * values to be used for deserializing Java enum values.
```

```
 * If set to 'false' numbers are acceptable and are used to map to
 * ordinal() of matching enumeration value; if 'true', numbers are
 * not allowed and a {@link JsonMappingException} will be thrown.
 * Latter behavior makes sense if there is concern that accidental
 * mapping from integer values to enums might happen (and when enums
 * are always serialized as JSON Strings)
 *<p>
 * Feature is disabled by default (to be consistent with behavior
 * of Jackson 1.6),
 * i.e. to allow use of JSON integers for Java enums.
 *
 * @since 1.7
 */
FAIL_ON_NUMBERS_FOR_ENUMS(false),

/**
 * Feature that determines whether Jackson code should catch
 * and wrap {@link Exception}s (but never {@link Error}s!)
 * to add additional information about
 * location (within input) of problem or not. If enabled,
 * most exceptions will be caught and re-thrown (exception
 * specifically being that {@link java.io.IOException}s may be passed
 * as is, since they are declared as throwable); this can be
 * convenient both in that all exceptions will be checked and
 * declared, and so there is more contextual information.
 * However, sometimes calling application may just want "raw"
 * unchecked exceptions passed as is.
 *<p>
 * Feature is enabled by default, and is similar in behavior
 * to default prior to 1.7.
 *
 * @since 1.7
 */
WRAP_EXCEPTIONS(true),

/*
/**********************************************************
 *  Structural conversion features
/**********************************************************
 */

/**
 * Feature that was planned to be enabled to handle "wrapped" values
 * (see {@link SerializationConfig.Feature#WRAP_ROOT_VALUE}
 * for details).
 * <b>NOTE</b>: Not implemented (unlike its counterpart for serialization
 * which was implemented in 1.7)
 *
```

```
     * @deprecated Never implemented; plus, incorrectly named: should be
     *    "UNWRAP_ROOT_VALUE" to be of use. Feature such named may be added in future.
     */
    @Deprecated
    WRAP_ROOT_VALUE(false)
    ;

    final boolean _defaultState;

    /**
     * Method that calculates bit set (flags) of all features that
     * are enabled by default.
     */
    public static int collectDefaults()
    {
        int flags = 0;
        for (Feature f : values()) {
            if (f.enabledByDefault()) {
                flags |= f.getMask();
            }
        }
        return flags;
    }

    private Feature(boolean defaultState) {
        _defaultState = defaultState;
    }

    public boolean enabledByDefault() { return _defaultState; }

    public int getMask() { return (1 << ordinal()); }
}

/**
 * Bitfield (set of flags) of all Features that are enabled
 * by default.
 */
protected final static int DEFAULT_FEATURE_FLAGS = Feature.collectDefaults();

protected final static DateFormat DEFAULT_DATE_FORMAT = StdDateFormat.instance;

/*
/**********************************************************
/* Configuration settings
/**********************************************************
 */

/**
```

```
 * Introspector used to figure out Bean properties needed for bean serialization
 * and deserialization. Overridable so that it is possible to change low-level
 * details of introspection, like adding new annotation types.
 */
protected ClassIntrospector<? extends BeanDescription> _classIntrospector;

/**
 * Introspector used for accessing annotation value based configuration.
 */
protected AnnotationIntrospector _annotationIntrospector;

/**
 * Bit set that contains all enabled features
 */
protected int _featureFlags = DEFAULT_FEATURE_FLAGS;

/**
 * Linked list that contains all registered problem handlers.
 * Implementation as front-added linked list allows for sharing
 * of the list (tail) without copying the list.
 */
protected LinkedNode<DeserializationProblemHandler> _problemHandlers;

/**
 * Custom date format to use for de-serialization. If specified, will be
 * used instead of {@link org.codehaus.jackson.map.util.StdDateFormat}.
 *<p>
 * Note that the configured format object will be cloned once per
 * deserialization process (first time it is needed)
 */
protected DateFormat _dateFormat = DEFAULT_DATE_FORMAT;

/**
 * Mapping that defines how to apply mix-in annotations: key is
 * the type to received additional annotations, and value is the
 * type that has annotations to "mix in".
 *<p>
 * Annotations associated with the value classes will be used to
 * override annotations of the key class, associated with the
 * same field or method. They can be further masked by sub-classes:
 * you can think of it as injecting annotations between the target
 * class and its sub-classes (or interfaces)
 *
 * @since 1.2
 */
protected HashMap<ClassKey,Class<?>> _mixInAnnotations;
```

```
/**
 * Flag used to detect when a copy if mix-in annotations is
 * needed: set when current copy is shared, cleared when a
 * fresh copy is made
 *
 * @since 1.2
 */
protected boolean _mixInAnnotationsShared;

/**
 * Type information handler used for "untyped" values (ones declared
 * to have type <code>Object.class</code>)
 *
 * @since 1.5
 */
protected final TypeResolverBuilder<?> _typer;

/**
 * Object used for determining whether specific property elements
 * (method, constructors, fields) can be auto-detected based on
 * their visibility (access modifiers). Can be changed to allow
 * different minimum visibility levels for auto-detection. Note
 * that this is the global handler; individual types (classes)
 * can further override active checker used (using
 * {@link JsonAutoDetect} annotation)
 *
 * @since 1.5
 */
protected VisibilityChecker<?> _visibilityChecker;

/**
 * Registered concrete subtypes that can be used instead of (or
 * in addition to) ones declared using annotations.
 *
 * @since 1.6
 */
protected SubtypeResolver _subtypeResolver;

/**
 * To support on-the-fly class generation for interface and abstract classes
 * it is possible to register "abstract type resolver".
 *
 * @since 1.6
 */
protected AbstractTypeResolver _abstractTypeResolver;

/**
 * Factory used for constructing {@link org.codehaus.jackson.JsonNode} instances.
```

```
 *
 * @since 1.6
 */
protected JsonNodeFactory _nodeFactory;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public DeserializationConfig(ClassIntrospector<? extends BeanDescription> intr,
                AnnotationIntrospector annIntr, VisibilityChecker<?> vc,
                SubtypeResolver subtypeResolver)
{
   _classIntrospector = intr;
   _annotationIntrospector = annIntr;
   _typer = null;
   _visibilityChecker = vc;
   _subtypeResolver = subtypeResolver;
   _nodeFactory = JsonNodeFactory.instance;
}

protected DeserializationConfig(DeserializationConfig src,
                    HashMap<ClassKey,Class<?>> mixins,
                    TypeResolverBuilder<?> typer,
                    VisibilityChecker<?> vc,
                    SubtypeResolver subtypeResolver)
{
   _classIntrospector = src._classIntrospector;
   _annotationIntrospector = src._annotationIntrospector;
   _abstractTypeResolver = src._abstractTypeResolver;
   _featureFlags = src._featureFlags;
   _problemHandlers = src._problemHandlers;
   _dateFormat = src._dateFormat;
   _nodeFactory = src._nodeFactory;
   _mixInAnnotations = mixins;
   _typer = typer;
   _visibilityChecker = vc;
   _subtypeResolver = subtypeResolver;
}

/*
/**********************************************************
/* Configuration: on/off features
/**********************************************************
 */
```

```
/**
 * Method for enabling specified  feature.
 */
public void enable(Feature f) {
    _featureFlags |= f.getMask();
}

/**
 * Method for disabling specified feature.
 */
public void disable(Feature f) {
    _featureFlags &= ~f.getMask();
}

/**
 * Method for enabling or disabling specified feature.
 */
public void set(Feature f, boolean state)
{
    if (state) {
        enable(f);
    } else {
        disable(f);
    }
}

/**
 * Method for checking whether given feature is enabled or not
 */
public final boolean isEnabled(Feature f) {
    return (_featureFlags & f.getMask()) != 0;
}

//protected int getFeatures() { return _generatorFeatures; }

/*
/**********************************************************
/* MapperConfig implementation
/**********************************************************
 */

/**
 * Method that checks class annotations that the argument Object has,
 * and modifies settings of this configuration object accordingly,
 * similar to how those annotations would affect actual value classes
 * annotated with them, but with global scope. Note that not all
 * annotations have global significance, and thus only subset of
 * Jackson annotations will have any effect.
```

```
 *<p>
 * Ones that are known to have effect are:
 *<ul>
 * <li>{@link JsonAutoDetect}</li>
 *</ul>
 *
 * @param cls Class of which class annotations to use
 *   for changing configuration settings
 */
//@Override
public void fromAnnotations(Class<?> cls)
{
 /* no class annotation for:
    *
    * - CAN_OVERRIDE_ACCESS_MODIFIERS
    * - USE_BIG_DECIMAL_FOR_FLOATS
    * - USE_BIG_INTEGER_FOR_INTS
    * - USE_GETTERS_AS_SETTERS
    */

   /* 10-Jul-2009, tatu: Should be able to just pass null as
    *    'MixInResolver'; no mix-ins set at this point
    */
   AnnotatedClass ac = AnnotatedClass.construct(cls, _annotationIntrospector, null);
   // visibility checks handled via separate checker object...
   _visibilityChecker = _annotationIntrospector.findAutoDetectVisibility(ac, _visibilityChecker);
}

/**
 * Method that is called to create a non-shared copy of the configuration
 * to be used for a deserialization operation.
 * Note that if sub-classing
 * and sub-class has additional instance methods,
 * this method <b>must</b> be overridden to produce proper sub-class
 * instance.
 */
//@Override
public DeserializationConfig createUnshared(TypeResolverBuilder<?> typer,
  VisibilityChecker<?> vc, SubtypeResolver subtypeResolver)
{
   HashMap<ClassKey,Class<?>> mixins = _mixInAnnotations;
   _mixInAnnotationsShared = true;
 return new DeserializationConfig(this, mixins, typer, vc, subtypeResolver);
}

/**
 * Alternative "copy factory" that creates an unshared copy that uses
 * different node factory than this instance.
```

```
     *
     * @since 1.6
     */
    public DeserializationConfig createUnshared(JsonNodeFactory nf)
    {
        DeserializationConfig config = createUnshared(_typer, _visibilityChecker, _subtypeResolver);
        config.setNodeFactory(nf);
        return config;
    }

    //@Override
    public void setIntrospector(ClassIntrospector<? extends BeanDescription> i) {
        _classIntrospector = i;
    }

    /**
     * Method for getting {@link AnnotationIntrospector} configured
     * to introspect annotation values used for configuration.
     */
    //@Override
    public AnnotationIntrospector getAnnotationIntrospector()
    {
        /* 29-Jul-2009, tatu: it's now possible to disable use of
         *   annotations; can be done using "no-op" introspector
         */
        if (isEnabled(Feature.USE_ANNOTATIONS)) {
            return _annotationIntrospector;
        }
        return NopAnnotationIntrospector.instance;
    }

    //@Override
    public void setAnnotationIntrospector(AnnotationIntrospector introspector)
    {
        _annotationIntrospector = introspector;
    }

    //@Override
    public void insertAnnotationIntrospector(AnnotationIntrospector introspector)
    {
        _annotationIntrospector = AnnotationIntrospector.Pair.create(introspector, _annotationIntrospector);
    }

    //@Override
    public void appendAnnotationIntrospector(AnnotationIntrospector introspector)
    {
        _annotationIntrospector = AnnotationIntrospector.Pair.create(_annotationIntrospector, introspector);
    }
```

```
/**
 * Method to use for defining mix-in annotations to use for augmenting
 * annotations that deserializable classes have.
 * Mixing in is done when introspecting class annotations and properties.
 * Map passed contains keys that are target classes (ones to augment
 * with new annotation overrides), and values that are source classes
 * (have annotations to use for augmentation).
 * Annotations from source classes (and their supertypes)
 * will <b>override</b>
 * annotations that target classes (and their super-types) have.
 *<p>
 * Note: a copy of argument Map is created; the original Map is
 * not modified or retained by this config object.
 *
 * @since 1.2
 */
//@Override
public void setMixInAnnotations(Map<Class<?>, Class<?>> sourceMixins)
{
   HashMap<ClassKey,Class<?>> mixins = null;
   if (sourceMixins != null && sourceMixins.size() > 0) {
      mixins = new HashMap<ClassKey,Class<?>>(sourceMixins.size());
      for (Map.Entry<Class<?>,Class<?>> en : sourceMixins.entrySet()) {
         mixins.put(new ClassKey(en.getKey()), en.getValue());
      }
   }
   _mixInAnnotationsShared = false;
   _mixInAnnotations = mixins;
}

//@Override
public void addMixInAnnotations(Class<?> target, Class<?> mixinSource)
{
   if (_mixInAnnotations == null || _mixInAnnotationsShared) {
      _mixInAnnotationsShared = false;
      _mixInAnnotations = new HashMap<ClassKey,Class<?>>();
   }
   _mixInAnnotations.put(new ClassKey(target), mixinSource);
}

/**
 * @since 1.2
 */
//@Override
public Class<?> findMixInClassFor(Class<?> cls) {
   return (_mixInAnnotations == null) ? null : _mixInAnnotations.get(new ClassKey(cls));
}
```

```
//@Override
public DateFormat getDateFormat() { return _dateFormat; }

/**
 * Method that will set the textual deserialization to use for
 * deserializing Dates (and Calendars). If null is passed, will
 * use {@link StdDateFormat}.
 */
//@Override
public void setDateFormat(DateFormat df) {
    _dateFormat = (df == null) ? StdDateFormat.instance : df;
}

//@Override
public VisibilityChecker<?> getDefaultVisibilityChecker() {
    return _visibilityChecker;
}

//@Override
public TypeResolverBuilder<?> getDefaultTyper(JavaType baseType) {
    return _typer;
}

/**
 * @since 1.6
 */
public SubtypeResolver getSubtypeResolver() {
    if (_subtypeResolver == null) {
        _subtypeResolver = new StdSubtypeResolver();
    }
    return _subtypeResolver;
}

/**
 * @since 1.6
 */
public void setSubtypeResolver(SubtypeResolver r) {
    _subtypeResolver = r;
}

/**
 * Accessor for getting bean description that only contains class
 * annotations: useful if no getter/setter/creator information is needed.
 *<p>
 * Note: part of {@link MapperConfig} since 1.7
 */
@SuppressWarnings("unchecked")
```

```
public <T extends BeanDescription> T introspectClassAnnotations(Class<?> cls) {
   return (T) _classIntrospector.forClassAnnotations(this, cls, this);
}

/**
 * Accessor for getting bean description that only contains immediate class
 * annotations: ones from the class, and its direct mix-in, if any, but
 * not from super types.
 *<p>
 * Note: part of {@link MapperConfig} since 1.7
 */
@SuppressWarnings("unchecked")
public <T extends BeanDescription> T introspectDirectClassAnnotations(Class<?> cls) {
   return (T) _classIntrospector.forDirectClassAnnotations(this, cls, this);
}

/*
/**********************************************************
/* Problem handlers
/**********************************************************
 */

/**
 * Method for getting head of the problem handler chain. May be null,
 * if no handlers have been added.
 */
public LinkedNode<DeserializationProblemHandler> getProblemHandlers()
{
   return _problemHandlers;
}

/**
 * Method that can be used to add a handler that can (try to)
 * resolve non-fatal deserialization problems.
 */
public void addHandler(DeserializationProblemHandler h)
{
   /* Sanity check: let's prevent adding same handler multiple
    * times
    */
   if (!LinkedNode.contains(_problemHandlers, h)) {
      _problemHandlers = new LinkedNode<DeserializationProblemHandler>(h, _problemHandlers);
   }
}

/**
 * Method for removing all configuring problem handlers; usually done to replace
 * existing handler(s) with different one(s)
```

```
     *
     * @since 1.1
     */
    public void clearHandlers()
    {
        _problemHandlers = null;
    }

    /*
    /**********************************************************
    /* Introspection methods
    /**********************************************************
     */

    /**
     * Method that will introspect full bean properties for the purpose
     * of building a bean deserializer
     *
     * @param type Type of class to be introspected
     */
    @SuppressWarnings("unchecked")
    public <T extends BeanDescription> T introspect(JavaType type) {
        return (T) _classIntrospector.forDeserialization(this, type, this);
    }

    /**
     * Method that will introspect subset of bean properties needed to
     * construct bean instance.
     */
    @SuppressWarnings("unchecked")
    public <T extends BeanDescription> T introspectForCreation(JavaType type) {
        return (T) _classIntrospector.forCreation(this, type, this);
    }

    /*
    /**********************************************************
    /* Polymorphic type handling configuration
    /**********************************************************
     */

    /**
     * Method for accessing {@link AbstractTypeResolver} configured, if any
     * (no default) used for resolving abstract types into concrete
     * types (either by mapping or materializing new classes).
     *
     * @since 1.6
     */
    public AbstractTypeResolver getAbstractTypeResolver() {
```

```
    return _abstractTypeResolver;
  }

  /**
   * @since 1.6
   */
  public void setAbstractTypeResolver(AbstractTypeResolver atr) {
    _abstractTypeResolver = atr;
  }

  /*
  /**********************************************************
  /* Other configuration
  /**********************************************************
   */

  /**
   * Method called during deserialization if Base64 encoded content
   * needs to be decoded. Default version just returns default Jackson
   * uses, which is modified-mime which does not add linefeeds (because
   * those would have to be escaped in Json strings).
   */
  public Base64Variant getBase64Variant() {
    return Base64Variants.getDefaultVariant();
  }

  /**
   * @since 1.6
   */
  public void setNodeFactory(JsonNodeFactory nf) {
    _nodeFactory = nf;
  }

  /**
   * @since 1.6
   */
  public final JsonNodeFactory getNodeFactory() {
    return _nodeFactory;
  }
}
/**
* Annotations that directly depend on Mapper classes (not just
* Jackson core) and are used for configuring Data Mapping functionality.
*/
package org.codehaus.jackson.map.annotate;
package org.codehaus.jackson.map.annotate;

import java.lang.annotation.ElementType;
```

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.codehaus.jackson.annotate.JacksonAnnotation;

/**
* Marker interface used to indicate implementation classes
* (serializers, deserializers etc) that are standard ones Jackson
* uses; not custom ones that application has added. It can be
* added in cases where certain optimizations can be made if
* default instances are uses; for example when handling conversions
* of "natural" JSON types like Strings, booleans and numbers.
*
* @since 1.6
*/
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JacksonStdImpl {

}
package org.codehaus.jackson.map.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.codehaus.jackson.annotate.JacksonAnnotation;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;

/**
* Annotation that can be used to explicitly define custom resolver
* used for handling serialization and deserialization of type information,
* needed for handling of polymorphic types (or sometimes just for linking
* abstract types to concrete types)
*
* @since 1.5
*/
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonTypeResolver {
  public Class<? extends TypeResolverBuilder<?>> value();
}
package org.codehaus.jackson.map.annotate;
```

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.codehaus.jackson.annotate.JacksonAnnotation;

/**
* Annotation used for indicating view(s) that the property
* that is defined by method or field annotated is part of.
*<p>
* An example annotation would be:
*<pre>
*  \@JsonView(BasicView.class)
*</pre>
* which would specify that property annotated would be included
* when processing (serializing, deserializing) View identified
* by <code>BasicView.class</code> (or its sub-class).
* If multiple View class identifiers are included, property will
* be part of all of them.
*
* @since 1.4
*/

@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonView {
  /**
   * View or views that annotated element is part of. Views are identified
   * by classes, and use expected class inheritance relationship: child
   * views contain all elements parent views have, for example.
   */
  public Class<?>[] value() default { };
}
package org.codehaus.jackson.map.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.codehaus.jackson.annotate.JacksonAnnotation;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;

/**
* Annotation that can be used to plug a custom type identifier handler
* ({@link TypeIdResolver})
```

```
* to be used by
* {@link org.codehaus.jackson.map.TypeSerializer}s
* and {@link org.codehaus.jackson.map.TypeDeserializer}s
* for converting between java types and type id included in JSON content.
*
* @author tatu
* @since 1.5
*/
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonTypeIdResolver
{
  public Class<? extends TypeIdResolver> value();
}
package org.codehaus.jackson.map.annotate;

/**
* Marker class used with annotations to indicate "no class". This is
* a silly but necessary work-around -- annotations can not take nulls
* as either default or explicit values. Hence for class values we must
* explicitly use a bogus placeholder to denote equivalent of
* "no class" (for which 'null' is usually the natural choice).
*<p>
* Note before version 1.4, this marker class was under
* "org.codehaus.jackson.annotate". However, since it is only used
* by annotations in "org.codehaus.jackson.map.annotate" (and not externally
* exposed), it was moved to that package as of version 1.5.
*/
public final class NoClass
{
  private NoClass() { }
}
package org.codehaus.jackson.map.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.codehaus.jackson.annotate.JacksonAnnotation;
import org.codehaus.jackson.map.JsonSerializer;

/**
* Annotation used for configuring serialization aspects, by attaching
* to "getter" methods or fields, or to value classes.
* When annotating value classes, configuration is used for instances
* of the value class but can be overridden by more specific annotations
```

```
* (ones that attach to methods or fields).
*<p>
* An example annotation would be:
*<pre>
*  &#64;JsonSerialize(using=MySerializer.class,
*    as=MySubClass.class,
*    include=JsonSerialize.Inclusion.NON_NULL,
*    typing=JsonSerialize.Typing.STATIC
*  )
*</pre>
* (which would be redundant, since some properties block others:
* specifically, 'using' has precedence over 'as', which has precedence
* over 'typing' setting)
*<p>
* NOTE: since version 1.2, annotation has also been applicable
* to (constructor) parameters
*
* @since 1.1
*/
@Target({ElementType.METHOD, ElementType.FIELD, ElementType.TYPE, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonSerialize
{
  // // // Annotations for explicitly specifying deserializer

  /**
   * Serializer class to use for
   * serializing associated value. Depending on what is annotated,
   * value is either an instance of annotated class (used globablly
   * anywhere where class serializer is needed); or only used for
   * serializing property access via a getter method.
   */
  public Class<? extends JsonSerializer<?>> using() default JsonSerializer.None.class;

  // // // Annotations for type handling, explicit declaration
  // // // (type used for choosing deserializer, if not explicitly
  // // // specified)

  /**
   * Subtype (of declared type, which itself is subtype of runtime type)
   * to use as type when locating serializer to use.
   *<p>
   * Bogus type {@link NoClass} can be used to indicate that declared
   * type is used as is (i.e. this annotation property has no setting);
   * this since annotation properties are not allowed to have null value.
   *<p>
   * Note: if {@link #using} is also used it has precedence
```

```
 * (since it directly specified
 * serializer, whereas this would only be used to locate the
 * serializer)
 * and value of this annotation property is ignored.
 */
public Class<?> as() default NoClass.class;

/**
 * Whether type detection used is dynamic or static: that is,
 * whether actual runtime type is used (dynamic), or just the
 * declared type (static).
 *
 * @since 1.2
 */
public Typing typing() default Typing.DYNAMIC;

// // // Annotation(s) for inclusion criteria

/**
 * Which properties of annotated Bean are
 * to be included in serialization (has no effect on other types
 * like enums, primitives or collections).
 * Choices are "all", "properties that have value other than null"
 * and "properties that have non-default value" (i.e. default value
 * being property setting for a Bean constructed with default no-arg
 * constructor, often null).
 *
 */
public Inclusion include() default Inclusion.ALWAYS;

/*
/**********************************************************
/* Value enumerations needed
/**********************************************************
 */

/**
 * Enumeration used with {@link JsonSerialize#include} property
 * to define which properties
 * of Java Beans are to be included in serialization
 */
public enum Inclusion
{
   /**
    * Value that indicates that properties are to be always included,
    * independent of value
    */
   ALWAYS,
```

```
    /**
     * Value that indicates that only properties with non-null
     * values are to be included.
     */
    NON_NULL,

    /**
     * Value that indicates that only properties that have values
     * that differ from default settings (meaning values they have
     * when Bean is constructed with its no-arguments constructor)
     * are to be included. Value is generally not useful with
     * {@link java.util.Map}s, since they have no default values;
     * and if used, works same as {@link #ALWAYS}.
     */
    NON_DEFAULT;
  }

  /**
   * Enumeration used with {@link JsonSerialize#typing} property
   * to define whether type detection is based on dynamic runtime
   * type (DYNAMIC) or declared type (STATIC).
   */
  public enum Typing
  {
    /**
     * Value that indicates that the actual dynamic runtime type is to
     * be used.
     */
    DYNAMIC,

    /**
     * Value that indicates that the static declared type is to
     * be used.
     */
    STATIC
    ;
  }
}
package org.codehaus.jackson.map.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.codehaus.jackson.annotate.JacksonAnnotation;
```

```
/**
* Annotation used to indicate which logical filter is to be used
* for filtering out properties of type (class) annotated;
* association made by this annotation declaring ids of filters,
* and  {@link org.codehaus.jackson.map.ObjectMapper} (or objects
* it delegates to) providing matching filters by id.
* Filters to use are of type
* {@link org.codehaus.jackson.map.ser.BeanPropertyFilter} and
* are registered through {@link org.codehaus.jackson.map.ObjectMapper}
*
* @since 1.7
*/
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonFilter
{
  /**
   * Id of filter to use; if empty String (""), no filter is to be used.
   */
  public String value();
}
package org.codehaus.jackson.map.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.codehaus.jackson.annotate.JacksonAnnotation;

/**
* Marker annotation used to denote whether given instance
* (currently only used with {@link org.codehaus.jackson.map.JsonDeserializer})
* can be cached.
*<p>
* Default action to take in absence of annotation depends
* on object using annotation; with deserializers default is
* to assume instances are not cachable.
*
* @since 1.1
*/
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonCachable
{
  /**
```

```
     * Default value is true, giving semantics for parameterless tag instance
     * such that empty instance indicates that instances of annotated class
     * are indeed cachable.
     */
    boolean value() default true;
}
package org.codehaus.jackson.map.annotate;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.codehaus.jackson.annotate.JacksonAnnotation;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.KeyDeserializer;

/**
* Annotation use for configuring deserialization aspects, by attaching
* to "setter" methods or fields, or to value classes.
* When annotating value classes, configuration is used for instances
* of the value class but can be overridden by more specific annotations
* (ones that attach to methods or fields).
*<p>
* An example annotation would be:
*<pre>
*  &#64;JsonDeserialize(using=MySerializer.class,
*    as=MyHashMap.class,
*    keyAs=MyHashKey.class,
*    contentAs=MyHashValue.class
*  )
*</pre>
*<p>
* NOTE: since version 1.2, annotation has also been applicable
* to (constructor) parameters
*
* @since 1.1
*/
@Target({ElementType.METHOD, ElementType.FIELD, ElementType.TYPE, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonDeserialize
{
  // // // Annotations for explicitly specifying deserializer

  /**
   * Deserializer class to use for deserializing associated value.
   * Depending on what is annotated,
```

```
 * value is either an instance of annotated class (used globablly
 * anywhere where class deserializer is needed); or only used for
 * deserializing property access via a setter method.
 */
public Class<? extends JsonDeserializer<?>> using()
    default JsonDeserializer.None.class;

/**
 * Deserializer class to use for deserializing contents (elements
 * of a Collection/array, values of Maps) of annotated property.
 * Can only be used on instances (methods, fields, constructors),
 * and not value classes themselves.
 *
 * @since 1.3
 */
public Class<? extends JsonDeserializer<?>> contentUsing()
    default JsonDeserializer.None.class;

/**
 * Deserializer class to use for deserializing Map keys
 * of annotated property.
 * Can only be used on instances (methods, fields, constructors),
 * and not value classes themselves.
 *
 * @since 1.3
 */
public Class<? extends KeyDeserializer> keyUsing()
    default KeyDeserializer.None.class;

// // // Annotations for explicitly specifying deserialization type
// // // (which is used for choosing deserializer, if not explicitly
// // // specified

/**
 * Concrete type to deserialize values as, instead of type otherwise
 * declared. Must be a subtype of declared type; otherwise an
 * exception may be thrown by deserializer.
 *<p>
 * Bogus type {@link NoClass} can be used to indicate that declared
 * type is used as is (i.e. this annotation property has no setting);
 * this since annotation properties are not allowed to have null value.
 *<p>
 * Note: if {@link #using} is also used it has precedence
 * (since it directly specified
 * deserializer, whereas this would only be used to locate the
 * deserializer)
 * and value of this annotation property is ignored.
 */
```

```
  public Class<?> as() default NoClass.class;

  /**
   * Concrete type to deserialize keys of {@link java.util.Map} as,
   * instead of type otherwise declared.
   * Must be a subtype of declared type; otherwise an exception may be
   * thrown by deserializer.
   *<p>
   * When annotating a class, will define default deserializer to
   * use when class instances are used as Map keys; when methods,
   * keys of the associated Map property.
   * Method annotation has precedence over class annotations, if both
   * exist.
   */
  public Class<?> keyAs() default NoClass.class;

  /**
   * Deserializer class to use for deserializing associated value
   * when it will used as contents of {@link java.util.Collection},
   * {@link java.util.Map} and array types.
   *<p>
   * When annotating a class, will define default deserializer to
   * use when class instances are used as contents of collection/map/array
   * types; when methods,
   * keys of the associated collection/map/array property.
   * Method annotation has precedence over class annotations, if both
   * exist.
   */
  public Class<?> contentAs() default NoClass.class;
}
package org.codehaus.jackson.map;

/**
* Interface used to indicate serializers that want to do post-processing
* after construction and being added to {@link SerializerProvider},
* but before being used. This is typically used to resolve references
* to other contained types; for example, bean serializers use this
* to eagerly find serializers for contained field types.
*/
public interface ResolvableSerializer
{
  /**
   * Method called after {@link SerializerProvider} has registered
   * the serializer, but before it has returned it to the caller.
   * Called object can then resolve its dependencies to other types,
   * including self-references (direct or indirect).
   *
   * @param provider Provider that has constructed serializer this method
```

```
     *   is called on.
     */
    public abstract void resolve(SerializerProvider provider)
        throws JsonMappingException;
}
package org.codehaus.jackson.map;

import org.codehaus.jackson.map.ser.BeanSerializerModifier;
import org.codehaus.jackson.type.JavaType;

/**
* Abstract class that defines API used by {@link SerializerProvider}
* to obtain actual
* {@link JsonSerializer} instances from multiple distinct factories.
*/
public abstract class SerializerFactory
{

    /*
    /**********************************************************
    /* Helper class to contain configuration settings
    /**********************************************************
     */

    /**
     * Configuration settings container class for bean serializer factory.
     *
     * @since 1.7
     */
    public abstract static class Config
    {
        /**
         * Method for creating a new instance with additional serializer provider.
         */
        public abstract Config withAdditionalSerializers(Serializers additional);

        /**
         * Method for creating a new instance with additional bean serializer modifier.
         */
        public abstract Config withSerializerModifier(BeanSerializerModifier modifier);

        public abstract boolean hasSerializers();

        public abstract boolean hasSerializerModifiers();

        public abstract Iterable<Serializers> serializers();

        public abstract Iterable<BeanSerializerModifier> serializerModifiers();
```

```
}

/*
/**********************************************************
/* Additional configuration
/**********************************************************
 */

/**
 * @since 1.7
 */
public abstract Config getConfig();

/**
 * Method used for creating a new instance of this factory, but with different
 * configuration. Reason for specifying factory method (instead of plain constructor)
 * is to allow proper sub-classing of factories.
 *<p>
 * Note that custom sub-classes generally <b>must override</b> implementation
 * of this method, as it usually requires instantiating a new instance of
 * factory type. Check out javadocs for
 * {@link org.codehaus.jackson.map.ser.BeanSerializerFactory} for more details.
 *
 * @since 1.7
 */
public abstract SerializerFactory withConfig(Config config);

/**
 * Convenience method for creating a new factory instance with additional serializer
 * provider; equivalent to calling
 *<pre>
 *   withConfig(getConfig().withAdditionalSerializers(additional));
 *<pre>
 *
 * @since 1.7
 */
public final SerializerFactory withAdditionalSerializers(Serializers additional) {
   return withConfig(getConfig().withAdditionalSerializers(additional));
}

/**
 * Convenience method for creating a new factory instance with additional bean
 * serializer modifier; equivalent to calling
 *<pre>
 *   withConfig(getConfig().withSerializerModifier(modifier));
 *<pre>
 *
 * @since 1.7
```

```
 */
public final SerializerFactory withSerializerModifier(BeanSerializerModifier modifier) {
   return withConfig(getConfig().withSerializerModifier(modifier));
}

/*
/**********************************************************
/* Basic SerializerFactory API:
/**********************************************************
 */

/**
 * Method called to create (or, for immutable serializers, reuse) a serializer for given type.
 */
public abstract JsonSerializer<Object> createSerializer(SerializationConfig config, JavaType baseType,
      BeanProperty property);

/**
 * Method called to create a type information serializer for given base type,
 * if one is needed. If not needed (no polymorphic handling configured), should
 * return null.
 *
 * @param baseType Declared type to use as the base type for type information serializer
 *
 * @return Type serializer to use for the base type, if one is needed; null if not.
 *
 * @since 1.7
 */
public abstract TypeSerializer createTypeSerializer(SerializationConfig config, JavaType baseType,
      BeanProperty property);

/*
/**********************************************************
/* Deprecated (as of 1.7) SerializerFactory API:
/**********************************************************
 */

/**
 * Deprecated version of accessor for type id serializer: as of 1.7 one needs
 * to instead call version that passes property information through.
 *
 * @since 1.5
 *
 * @deprecated Since 1.7, call variant with more arguments
 */
@Deprecated
public final JsonSerializer<Object> createSerializer(JavaType type, SerializationConfig config) {
   return createSerializer(config, type, null);
```

```
    }

    /**
     * Deprecated version of accessor for type id serializer: as of 1.7 one needs
     * to instead call version that passes property information through.
     *
     * @since 1.5
     *
     * @deprecated Since 1.7, call variant with more arguments
     */
    @Deprecated
    public final TypeSerializer createTypeSerializer(JavaType baseType, SerializationConfig config) {
        return createTypeSerializer(config, baseType, null);
    }
}
package org.codehaus.jackson.map;

import java.io.IOException;

import org.codehaus.jackson.JsonProcessingException;

/**
* This is the class that can be registered (via
* {@link DeserializationConfig} object owner by
* {@link ObjectMapper}) to get calledn when a potentially
* recoverable problem is encountered during deserialization
* process. Handlers can try to resolve the problem, throw
* an exception or do nothing.
*<p>
* Default implementations for all methods implemented minimal
* "do nothing" functionality, which is roughly equivalent to
* not having a registered listener at all. This allows for
* only implemented handler methods one is interested in, without
* handling other cases.
*
* @author tatu
*/
public abstract class DeserializationProblemHandler
{
    /**
     * Method called when a Json Map ("Object") entry with an unrecognized
     * name is encountered.
     * Content (supposedly) matching the property are accessible via
     * parser that can be obtained from passed deserialization context.
     * Handler can also choose to skip the content; if so, it MUST return
     * true to indicate it did handle property succesfully.
     * Skipping is usually done like so:
     *<pre>
```

```
   *  ctxt.getParser().skipChildren();
   *</pre>
   *<p>
   * Note: version 1.2 added new deserialization feature
   * (<code>DeserializationConfig.Feature.FAIL_ON_UNKNOWN_PROPERTIES</code>).
   * It will only have effect <b>after</b> handler is called, and only
   * if handler did <b>not</b> handle the problem.
   *
   * @param beanOrClass Either bean instance being deserialized (if one
   *   has been instantiated so far); or Class that indicates type that
   *   will be instantiated (if no instantiation done yet: for example
   *   when bean uses non-default constructors)
   *
   * @return True if the problem was succesfully resolved (and content available
   *    used or skipped); false if listen
   */
  public boolean handleUnknownProperty(DeserializationContext ctxt, JsonDeserializer<?> deserializer,
                                       Object beanOrClass, String propertyName)
    throws IOException, JsonProcessingException
  {
    return false;
  }
}
package org.codehaus.jackson.map;

import org.codehaus.jackson.type.JavaType;

/**
* Interface that defines API for simple extensions that can provide additional serializers
* for various types. Access is by a single callback method; instance is to either return
* a configured {@link JsonSerializer} for specified type, or null to indicate that it
* does not support handling of the type. In latter case, further calls can be made
* for other providers; in former case returned serializer is used for handling of
* instances of specified type.
*
* @since 1.7
*/
public interface Serializers
{
  /**
   * Method called serialization framework first time a serializer is needed for
   * specified type. Implementation should return a serializer instance if it supports
   * specified type; or null if it does not.
   *
   * @param type Fully resolved type of instances to serialize
   * @param config Serialization configuration in use
   * @param beanDesc Additional information about type; will always be of type
   *    {@link org.codehaus.jackson.map.introspect.BasicBeanDescription} (that is,
```

```
     *    safe to cast to this more specific type)
     * @param property Property that contains values to serialize
     *
     * @return Configured serializer to use for the type; or null if implementation
     *    does not recognize or support type
     */
    public JsonSerializer<?> findSerializer(SerializationConfig config,
            JavaType type, BeanDescription beanDesc, BeanProperty property);
}
package org.codehaus.jackson.map;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.annotate.JsonTypeInfo.As;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;

/**
* Interface for deserializing type information from JSON content, to
* type-safely deserialize data into correct polymorphic instance
* (when type inclusion has been enabled for type handled).
*<p>
* Separate deserialization methods are needed because serialized
* form for inclusion mechanism {@link As#PROPERTY}
* is slighty different if value is not expressed as JSON Object:
* and as such both type deserializer and serializer need to
* JSON Object form (array, object or other (== scalar)) being
* used.
*
* @since 1.5
* @author tatus
*/
public abstract class TypeDeserializer
{
    /*
    /**********************************************************
    /* Introspection
    /**********************************************************
     */

    /**
     * Accessor for type information inclusion method
     * that deserializer uses; indicates how type information
     * is (expected to be) embedded in JSON input.
     */
    public abstract As getTypeInclusion();

    /**
```

```
     * Name of property that contains type information, if
     * property-based inclusion is used.
     */
    public abstract String getPropertyName();

    /**
     * Accessor for object that handles conversions between
     * types and matching type ids.
     */
    public abstract TypeIdResolver getTypeIdResolver();

    /*
    /**********************************************************
    /* Type deserialization methods
    /**********************************************************
     */

    /**
     * Method called to let this type deserializer handle
     * deserialization of "typed" object, when value itself
     * is serialized as JSON Object (regardless of Java type).
     * Method needs to figure out intended
     * polymorphic type, locate {@link JsonDeserializer} to use, and
     * call it with JSON data to deserializer (which does not contain
     * type information).
     */
    public abstract Object deserializeTypedFromObject(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException;

    /**
     * Method called to let this type deserializer handle
     * deserialization of "typed" object, when value itself
     * is serialized as JSON Array (regardless of Java type).
     * Method needs to figure out intended
     * polymorphic type, locate {@link JsonDeserializer} to use, and
     * call it with JSON data to deserializer (which does not contain
     * type information).
     */
    public abstract Object deserializeTypedFromArray(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException;

    /**
     * Method called to let this type deserializer handle
     * deserialization of "typed" object, when value itself
     * is serialized as a scalar JSON value (something other
     * than Array or Object), regardless of Java type.
     * Method needs to figure out intended
     * polymorphic type, locate {@link JsonDeserializer} to use, and
```

```
     * call it with JSON data to deserializer (which does not contain
     * type information).
     */
    public abstract Object deserializeTypedFromScalar(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException;

    /**
     * Method called to let this type deserializer handle
     * deserialization of "typed" object, when value itself
     * may have been serialized using any kind of JSON value
     * (Array, Object, scalar). Should only be called if JSON
     * serialization is polymorphic (not Java type); for example when
     * using JSON node representation, or "untyped" Java object
     * (which may be Map, Collection, wrapper/primitive etc).
     */
    public abstract Object deserializeTypedFromAny(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException;

}

package org.codehaus.jackson.map;

import java.io.Serializable;
import java.util.*;

import org.codehaus.jackson.*;

/**
* Checked exception used to signal fatal problems with mapping of
* content.
*<p>
* One additional feature is the ability to denote relevant path
* of references (during serialization/deserialization) to help in
* troubleshooting.
*/
public class JsonMappingException
    extends JsonProcessingException
{
    private static final long serialVersionUID = 1L;

    /**
     * Let's limit length of reference chain, to limit damage in cases
     * of infinite recursion.
     */
    final static int MAX_REFS_TO_LIST = 1000;

    /*
    /**********************************************************
```

```
/* Helper classes
/**********************************************************
 */

/**
 * Simple bean class used to contain references. References
 * can be added to indicate execution/reference path that
 * lead to the problem that caused this exception to be
 * thrown.
 */
public static class Reference implements Serializable
{
    private static final long serialVersionUID = 1L;

    /**
     * Object through which reference was resolved. Can be either
     * actual instance (usually the case for serialization), or
     * Class (usually the case for deserialization).
     */
    protected Object _from;

    /**
     * Name of field (for beans) or key (for Maps) that is part
     * of the reference. May be null for Collection types (which
     * generally have {@link #_index} defined), or when resolving
     * Map classes without (yet) having an instance to operate on.
     */
    protected String _fieldName;

    /**
     * Index within a {@link Collection} instance that contained
     * the reference; used if index is relevant and available.
     * If either not applicable, or not available, -1 is used to
     * denote "not known".
     */
    protected int _index = -1;

    /**
     * Default constructor for deserialization/sub-classing purposes
     */
    protected Reference() { }

    public Reference(Object from) { _from = from; }

    public Reference(Object from, String fieldName) {
        _from = from;
        if (fieldName == null) {
            throw new NullPointerException("Can not pass null fieldName");
```

```
        }
        _fieldName = fieldName;
    }

    public Reference(Object from, int index) {
        _from = from;
        _index = index;
    }

    public void setFrom(Object o) { _from = o; }
    public void setFieldName(String n) { _fieldName = n; }
    public void setIndex(int ix) { _index = ix; }

    public Object getFrom() { return _from; }
    public String getFieldName() { return _fieldName; }
    public int getIndex() { return _index; }

    @Override public String toString() {
        StringBuilder sb = new StringBuilder();
        Class<?> cls = (_from instanceof Class<?>) ?
            ((Class<?>)_from) : _from.getClass();
        /* Hmmh. Although Class.getName() is mostly ok, it does look
         * butt-ugly for arrays. So let's use getSimpleName() instead;
         * but have to prepend package name too.
         */
        Package pkg = cls.getPackage();
        if (pkg != null) {
            sb.append(pkg.getName());
            sb.append('.');
        }
        sb.append(cls.getSimpleName());
        sb.append('[');
        if (_fieldName != null) {
            sb.append("'");
            sb.append(_fieldName);
            sb.append("'");
        } else if (_index >= 0) {
            sb.append(_index);
        } else {
            sb.append('?');
        }
        sb.append(']');
        return sb.toString();
    }
}

/*
/**********************************************************
```

```
/* State/configuration
/**********************************************************
 */

/**
 * Path through which problem that triggering throwing of
 * this exception was reached.
 */
protected LinkedList<Reference> _path;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public JsonMappingException(String msg)
{
   super(msg);
}

public JsonMappingException(String msg, Throwable rootCause)
{
   super(msg, rootCause);
}

public JsonMappingException(String msg, JsonLocation loc)
{
   super(msg, loc);
}

public JsonMappingException(String msg, JsonLocation loc, Throwable rootCause)
{
   super(msg, loc, rootCause);
}

public static JsonMappingException from(JsonParser jp, String msg)
{
   return new JsonMappingException(msg, jp.getTokenLocation());
}

public static JsonMappingException from(JsonParser jp, String msg,
                                        Throwable problem)
{
   return new JsonMappingException(msg, jp.getTokenLocation(), problem);
}

/**
```

```
     * Method that can be called to either create a new JsonMappingException
     * (if underlying exception is not a JsonMappingException), or augment
     * given exception with given path/reference information.
     *
     * This version of method is called when the reference is through a
     * non-indexed object, such as a Map or POJO/bean.
     */
    public static JsonMappingException wrapWithPath(Throwable src, Object refFrom,
                                        String refFieldName)
    {
       return wrapWithPath(src, new Reference(refFrom, refFieldName));
    }

    /**
     * Method that can be called to either create a new JsonMappingException
     * (if underlying exception is not a JsonMappingException), or augment
     * given exception with given path/reference information.
     *
     * This version of method is called when the reference is through an
     * index, which happens with arrays and Collections.
     */
    public static JsonMappingException wrapWithPath(Throwable src, Object refFrom,
                                        int index)
    {
       return wrapWithPath(src, new Reference(refFrom, index));
    }

    /**
     * Method that can be called to either create a new JsonMappingException
     * (if underlying exception is not a JsonMappingException), or augment
     * given exception with given path/reference information.
     */
    public static JsonMappingException wrapWithPath(Throwable src, Reference ref)
    {
       JsonMappingException jme;
       if (src instanceof JsonMappingException) {
          jme = (JsonMappingException) src;
       } else {
          String msg = src.getMessage();
          /* Related to [JACKSON-62], let's use a more meaningful placeholder
           * if all we have is null
           */
          if (msg == null || msg.length() == 0) {
             msg = "(was "+src.getClass().getName()+")";
          }
          jme = new JsonMappingException(msg, null, src);
       }
       jme.prependPath(ref);
```

```
    return jme;
}

/*
/**********************************************************
/* Accessors/mutators
/**********************************************************
 */

public List<Reference> getPath()
{
    if (_path == null) {
        return Collections.emptyList();
    }
    return Collections.unmodifiableList(_path);
}

/**
 * Method called to prepend a reference information in front of
 * current path
 */
public void prependPath(Object referrer, String fieldName)
{
    Reference ref = new Reference(referrer, fieldName);
    prependPath(ref);
}
/**
 * Method called to prepend a reference information in front of
 * current path
 */
public void prependPath(Object referrer, int index)
{
    Reference ref = new Reference(referrer, index);
    prependPath(ref);
}

public void prependPath(Reference r)
{
    if (_path == null) {
        _path = new LinkedList<Reference>();
    }
    /* Also: let's not increase without bounds. Could choose either
     * head or tail; tail is easier (no need to ever remove), as
     * well as potentially more useful so let's use it:
     */
    if (_path.size() < MAX_REFS_TO_LIST) {
        _path.addFirst(r);
    }
```

```
}

/*
/**********************************************************
/* Overridden methods
/**********************************************************
 */

/**
 * Method is overridden so that we can properly inject description
 * of problem path, if such is defined.
 */
@Override
public String getMessage()
{
   /* First: if we have no path info, let's just use parent's
    * definition as is
    */
   String msg = super.getMessage();
   if (_path == null) {
      return msg;
   }
   /* 19-Feb-2009, tatu: Null and empty messages are not very
    *   useful (plus nulls would lead to NPEs), so let's
    *   use something else
    */
   StringBuilder sb = (msg == null) ? new StringBuilder() : new StringBuilder(msg);
   /* 18-Feb-2009, tatu: initially there was a linefeed between
    *    message and path reference; but unfortunately many systems
    *   (loggers, junit) seem to assume linefeeds are only added to
    *   separate stack trace.
    */
   sb.append(" (through reference chain: ");
   _appendPathDesc(sb);
   sb.append(')');
   return sb.toString();
}

@Override
public String toString()
{
   return getClass().getName()+": "+getMessage();
}

/*
/**********************************************************
/* Internal methods
/**********************************************************
```

```
     */

    protected void _appendPathDesc(StringBuilder sb)
    {
        Iterator<Reference> it = _path.iterator();
        while (it.hasNext()) {
            sb.append(it.next().toString());
            if (it.hasNext()) {
                sb.append("->");
            }
        }
    }
}
package org.codehaus.jackson.map;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.annotate.JsonTypeInfo;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;

/**
* Interface for serializing type information regarding instances of specified
* base type (super class), so that exact subtype can be properly deserialized
* later on. These instances are to be called by regular
* {@link org.codehaus.jackson.map.JsonSerializer}s using proper contextual
* calls, to add type information using mechanism type serializer was
* configured with.
*
* @since 1.5
* @author tatus
*/
public abstract class TypeSerializer
{
    /*
     ***********************************************************
     * Introspection
     ***********************************************************
     */

    /**
     * Accessor for type information inclusion method
     * that serializer uses; indicates how type information
     * is embedded in resulting JSON.
     */
    public abstract JsonTypeInfo.As getTypeInclusion();

    /**
```

```
     * Name of property that contains type information, if
     * property-based inclusion is used.
     */
    public abstract String getPropertyName();

    /**
     * Accessor for object that handles conversions between
     * types and matching type ids.
     */
    public abstract TypeIdResolver getTypeIdResolver();

    /*
    /**********************************************************
    * Type serialization methods
    /**********************************************************
    */

    /**
     * Method called to write initial part of type information for given
     * value, when it will be output as scalar JSON value (not as JSON
     * Object or Array).
     * This means that the context after call can not be that of JSON Object;
     * it may be Array or root context.
     *
     * @param value Value that will be serialized, for which type information is
     *   to be written
     * @param jgen Generator to use for writing type information
     */
    public abstract void writeTypePrefixForScalar(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException;

    /**
     * Method called to write initial part of type information for given
     * value, when it will be output as JSON Object value (not as JSON
     * Array or scalar).
     * This means that context after call must be JSON Object, meaning that
     * caller can then proceed to output field entries.
     *
     * @param value Value that will be serialized, for which type information is
     *   to be written
     * @param jgen Generator to use for writing type information
     */
    public abstract void writeTypePrefixForObject(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException;

    /**
     * Method called to write initial part of type information for given
     * value, when it will be output as JSON Array value (not as JSON
```

```
     * Object or scalar).
     * This means that context after call must be JSON Array, that is, there
     * must be an open START_ARRAY to write contents in.
     *
     * @param value Value that will be serialized, for which type information is
     *   to be written
     * @param jgen Generator to use for writing type information
     */
    public abstract void writeTypePrefixForArray(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException;

    /**
     * Method called after value has been serialized, to close any scopes opened
     * by earlier matching call to {@link #writeTypePrefixForScalar}.
     * It needs to write closing END_OBJECT marker, and any other decoration
     * that needs to be matched.
     */
    public abstract void writeTypeSuffixForScalar(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException;

    /**
     * Method called after value has been serialized, to close any scopes opened
     * by earlier matching call to {@link #writeTypePrefixForObject}.
     * It needs to write closing END_OBJECT marker, and any other decoration
     * that needs to be matched.
     */
    public abstract void writeTypeSuffixForObject(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException;

    public abstract void writeTypeSuffixForArray(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException;
}
/**
* Contains implementation classes of deserialization part of
* data binding.
*/
package org.codehaus.jackson.map.deser;
package org.codehaus.jackson.map.deser;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.node.*;

/**
* Deserializer that can build instances of {@link JsonNode} from any
* JSON content, using appropriate {@link JsonNode} type.
```

```
*/
public class JsonNodeDeserializer
  extends BaseNodeDeserializer<JsonNode>
{
  /**
   * Singleton instance of generic deserializer for {@link JsonNode}
   *
   * @deprecated Use {@link #getDeserializer} accessor instead
   */
  @Deprecated
  public final static JsonNodeDeserializer instance = new JsonNodeDeserializer();

  protected JsonNodeDeserializer() { super(JsonNode.class); }

  /**
   * Factory method for accessing deserializer for specific node type
   */
  public static JsonDeserializer<? extends JsonNode> getDeserializer(Class<?> nodeClass)
  {
    if (nodeClass == ObjectNode.class) {
      return ObjectDeserializer.getInstance();
    }
    if (nodeClass == ArrayNode.class) {
      return ArrayDeserializer.getInstance();
    }
    // For others, generic one works fine
    return instance;
  }

  /*
  /**********************************************************
  /* Actual deserializer implementations
  /**********************************************************
   */

  /**
   * Implementation that will produce types of any JSON nodes; not just one
   * deserializer is registered to handle (in case of more specialized handler).
   * Overridden by typed sub-classes for more thorough checking
   */
  @Override
  public JsonNode deserialize(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
    return deserializeAny(jp, ctxt);
  }

  /*
```

```
/**********************************************************
/* Specific instances for more accurate types
/**********************************************************
 */

final static class ObjectDeserializer
   extends BaseNodeDeserializer<ObjectNode>
{
   protected final static ObjectDeserializer _instance = new ObjectDeserializer();

   protected ObjectDeserializer() {
      super(ObjectNode.class);
   }

   public static ObjectDeserializer getInstance() { return _instance; }

   @Override
   public ObjectNode deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException
   {
      if (jp.getCurrentToken() == JsonToken.START_OBJECT) {
         jp.nextToken();
         return deserializeObject(jp, ctxt);
      }
      if (jp.getCurrentToken() == JsonToken.FIELD_NAME) {
         return deserializeObject(jp, ctxt);
      }
      throw ctxt.mappingException(ObjectNode.class);
    }
}

final static class ArrayDeserializer
   extends BaseNodeDeserializer<ArrayNode>
{
   protected final static ArrayDeserializer _instance = new ArrayDeserializer();

   protected ArrayDeserializer() {
      super(ArrayNode.class);
   }

   public static ArrayDeserializer getInstance() { return _instance; }

   @Override
   public ArrayNode deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException
   {
      if (jp.isExpectedStartArrayToken()) {
         return deserializeArray(jp, ctxt);
```

```
            }
            throw ctxt.mappingException(ArrayNode.class);
        }
    }
}

/**
* Base class for all actual {@link JsonNode} deserializer
* implementations
*/
abstract class BaseNodeDeserializer<N extends JsonNode>
    extends StdDeserializer<N>
{
    public BaseNodeDeserializer(Class<N> nodeClass)
    {
        super(nodeClass);
    }

    @Override
    public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
            TypeDeserializer typeDeserializer)
        throws IOException, JsonProcessingException
    {
        /* Output can be as JSON Object, Array or scalar: no way to know
         * a priori. So:
         */
        return typeDeserializer.deserializeTypedFromAny(jp, ctxt);
    }

    /*
    /**********************************************************
    /* Overridable methods
    /**********************************************************
     */

    protected void _reportProblem(JsonParser jp, String msg)
        throws JsonMappingException
    {
        throw new JsonMappingException(msg, jp.getTokenLocation());
    }

    /**
     * Method called when there is a duplicate value for a field.
     * By default we don't care, and the last value is used.
     * Can be overridden to provide alternate handling, such as throwing
     * an exception, or choosing different strategy for combining values
     * or choosing which one to keep.
     *
```

```
     * @param fieldName Name of the field for which duplicate value was found
     * @param objectNode Object node that contains values
     * @param oldValue Value that existed for the object node before newValue
     *   was added
     * @param newValue Newly added value just added to the object node
     */
    protected void _handleDuplicateField(String fieldName, ObjectNode objectNode,
                                         JsonNode oldValue, JsonNode newValue)
        throws JsonProcessingException
    {
        // By default, we don't do anything
        ;
    }

    /*
    /**********************************************************
    /* Helper methods
    /**********************************************************
     */

    protected final ObjectNode deserializeObject(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        ObjectNode node = ctxt.getNodeFactory().objectNode();
        JsonToken t = jp.getCurrentToken();
        if (t == JsonToken.START_OBJECT) {
            t = jp.nextToken();
        }
        for (; t == JsonToken.FIELD_NAME; t = jp.nextToken()) {
            String fieldName = jp.getCurrentName();
            jp.nextToken();
            JsonNode value = deserializeAny(jp, ctxt);
            JsonNode old = node.put(fieldName, value);
            if (old != null) {
                _handleDuplicateField(fieldName, node, old, value);
            }
        }
        return node;
    }

    protected final ArrayNode deserializeArray(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        ArrayNode node = ctxt.getNodeFactory().arrayNode();
        while (jp.nextToken() != JsonToken.END_ARRAY) {
            node.add(deserializeAny(jp, ctxt));
        }
        return node;
```

```
}

protected final JsonNode deserializeAny(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   final JsonNodeFactory nodeFactory = ctxt.getNodeFactory();
   switch (jp.getCurrentToken()) {
   case START_OBJECT:
   case FIELD_NAME:
      return deserializeObject(jp, ctxt);

   case START_ARRAY:
      return deserializeArray(jp, ctxt);

   case VALUE_STRING:
      return nodeFactory.textNode(jp.getText());

   case VALUE_NUMBER_INT:
      {
         JsonParser.NumberType nt = jp.getNumberType();
         if (nt == JsonParser.NumberType.BIG_INTEGER
            || ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_INTEGER_FOR_INTS)) {
            return nodeFactory.numberNode(jp.getBigIntegerValue());
         }
         if (nt == JsonParser.NumberType.INT) {
            return nodeFactory.numberNode(jp.getIntValue());
         }
         return nodeFactory.numberNode(jp.getLongValue());
      }

   case VALUE_NUMBER_FLOAT:
      {
         JsonParser.NumberType nt = jp.getNumberType();
         if (nt == JsonParser.NumberType.BIG_DECIMAL
            || ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_DECIMAL_FOR_FLOATS)) {
            return nodeFactory.numberNode(jp.getDecimalValue());
         }
         return nodeFactory.numberNode(jp.getDoubleValue());
      }

   case VALUE_TRUE:
      return nodeFactory.booleanNode(true);

   case VALUE_FALSE:
      return nodeFactory.booleanNode(false);

   case VALUE_NULL:
      return nodeFactory.nullNode();
```

```
            // These states can not be mapped; input stream is
            // off by an event or two

        case END_OBJECT:
        case END_ARRAY:
        default:
            throw ctxt.mappingException(getValueClass());
        }
    }
}
package org.codehaus.jackson.map.deser;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.util.*;

import org.codehaus.jackson.map.DeserializationConfig;
import org.codehaus.jackson.map.KeyDeserializer;
import org.codehaus.jackson.map.type.*;
import org.codehaus.jackson.map.introspect.BasicBeanDescription;
import org.codehaus.jackson.type.JavaType;

/**
* Helper class used to contain simple/well-known key deserializers.
* Following kinds of Objects can be handled currently:
*<ul>
* <li>Primitive wrappers</li>
* <li>Enums (usually not needed, since EnumMap doesn't call us)</li>
* <li>Anything with constructor that takes a single String arg
*   (if not explicitly @JsonIgnore'd)</li>
* <li>Anything with 'static T valueOf(String)' factory method
*   (if not explicitly @JsonIgnore'd)</li>
*</ul>
*/
class StdKeyDeserializers
{
    final HashMap<JavaType, KeyDeserializer> _keyDeserializers = new HashMap<JavaType, KeyDeserializer>();

    private StdKeyDeserializers()
    {
        add(new StdKeyDeserializer.BoolKD());
        add(new StdKeyDeserializer.ByteKD());
        add(new StdKeyDeserializer.CharKD());
        add(new StdKeyDeserializer.ShortKD());
        add(new StdKeyDeserializer.IntKD());
        add(new StdKeyDeserializer.LongKD());
        add(new StdKeyDeserializer.FloatKD());
```

```
        add(new StdKeyDeserializer.DoubleKD());
    }

    private void add(StdKeyDeserializer kdeser)
    {
        Class<?> keyClass = kdeser.getKeyClass();
        _keyDeserializers.put(TypeFactory.type(keyClass), kdeser);
    }

    public static HashMap<JavaType, KeyDeserializer> constructAll()
    {
        return new StdKeyDeserializers()._keyDeserializers;
    }

    /*
    /**********************************************************
    /* Dynamic factory methods
    /**********************************************************
     */

    public static KeyDeserializer constructEnumKeyDeserializer(DeserializationConfig config, JavaType type)
    {
        EnumResolver<?> er = EnumResolver.constructUnsafe(type.getRawClass(),
    config.getAnnotationIntrospector());
        return new StdKeyDeserializer.EnumKD(er);
    }

    public static KeyDeserializer findStringBasedKeyDeserializer(DeserializationConfig config, JavaType type)
    {
        /* We don't need full deserialization information, just need to
         * know creators.
         */
     BasicBeanDescription beanDesc = config.introspect(type);
        // Ok, so: can we find T(String) constructor?
        Constructor<?> ctor = beanDesc.findSingleArgConstructor(String.class);
        if (ctor != null) {
            return new StdKeyDeserializer.StringCtorKeyDeserializer(ctor);
        }
        /* or if not, "static T valueOf(String)" (or equivalent marked
         * with @JsonCreator annotation?)
         */
        Method m = beanDesc.findFactoryMethod(String.class);
        if (m != null){
            return new StdKeyDeserializer.StringFactoryKeyDeserializer(m);
        }
        // nope, no such luck...
        return null;
    }
```

```
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.util.*;

import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.TypeDeserializer;

/**
*
* <p>
* Note: casting within this class is all messed up -- just could not figure out a way
* to properly deal with recursive definition of "EnumMap<K extends Enum<K>, V>
*
* @author tsaloranta
*/
public final class EnumMapDeserializer
  extends StdDeserializer<EnumMap<?,?>>
{
  final EnumResolver<?> _enumResolver;

  final JsonDeserializer<Object> _valueDeserializer;

  public EnumMapDeserializer(EnumResolver<?> enumRes, JsonDeserializer<Object> valueDes)
  {
    super(EnumMap.class);
    _enumResolver = enumRes;
    _valueDeserializer = valueDes;
  }

  @SuppressWarnings("unchecked")
  @Override
  public EnumMap<?,?> deserialize(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
    // Ok: must point to START_OBJECT
    if (jp.getCurrentToken() != JsonToken.START_OBJECT) {
      throw ctxt.mappingException(EnumMap.class);
    }
    EnumMap result = constructMap();

    while ((jp.nextToken()) != JsonToken.END_OBJECT) {
      String fieldName = jp.getCurrentName();
```

```
            Enum<?> key = _enumResolver.findEnum(fieldName);
            if (key == null) {
                throw ctxt.weirdStringException(_enumResolver.getEnumClass(), "value not one of declared Enum
instance names");
            }
            // And then the value...
            JsonToken t = jp.nextToken();
            /* note: MUST check for nulls separately: deserializers will
             * not handle them (and maybe fail or return bogus data)
             */
            Object value = (t == JsonToken.VALUE_NULL) ?
                null :  _valueDeserializer.deserialize(jp, ctxt);
            result.put(key, value);
        }
        return result;
    }

    @Override
    public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
            TypeDeserializer typeDeserializer)
        throws IOException, JsonProcessingException
    {
        // In future could check current token... for now this should be enough:
        return typeDeserializer.deserializeTypedFromObject(jp, ctxt);
    }

    @SuppressWarnings("unchecked")
    private EnumMap<?,?> constructMap()
    {
        Class<? extends Enum<?>> enumCls = _enumResolver.getEnumClass();
     return new EnumMap(enumCls);
    }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.util.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.type.*;
import org.codehaus.jackson.map.util.ArrayBuilders;
import org.codehaus.jackson.map.util.ObjectBuffer;
import org.codehaus.jackson.type.JavaType;

/**
* Container for deserializers used for instantiating "primitive arrays",
```

```
* arrays that contain non-object java primitive types.
*/
public class ArrayDeserializers
{
  HashMap<JavaType,JsonDeserializer<Object>> _allDeserializers;

  final static ArrayDeserializers instance = new ArrayDeserializers();

  private ArrayDeserializers()
  {
    _allDeserializers = new HashMap<JavaType,JsonDeserializer<Object>>();
    // note: we'll use component type as key, not array type
    add(boolean.class, new BooleanDeser());

    /* ByteDeser is bit special, as it has 2 separate modes of operation;
     * one for String input (-> base64 input), the other for
     * numeric input
     */
    add(byte.class, new ByteDeser());
    add(short.class, new ShortDeser());
    add(int.class, new IntDeser());
    add(long.class, new LongDeser());

    add(float.class, new FloatDeser());
    add(double.class, new DoubleDeser());

    add(String.class, new StringDeser());
    /* also: char[] is most likely only used with Strings; doesn't
     * seem to make sense to transfer as numbers
     */
    add(char.class, new CharDeser());
  }

  public static HashMap<JavaType,JsonDeserializer<Object>> getAll()
  {
    return instance._allDeserializers;
  }

  @SuppressWarnings("unchecked")
 private void add(Class<?> cls, JsonDeserializer<?> deser)
  {
    _allDeserializers.put(TypeFactory.type(cls),
                          (JsonDeserializer<Object>) deser);
  }

  public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
         TypeDeserializer typeDeserializer)
    throws IOException, JsonProcessingException
```

```
{
    /* Should there be separate handling for base64 stuff?
     * for now this should be enough:
     */
    return typeDeserializer.deserializeTypedFromArray(jp, ctxt);
}

/*
/**********************************************************
/* Intermediate base class
/**********************************************************
 */

static abstract class ArrayDeser<T>
    extends StdDeserializer<T>
{
    protected ArrayDeser(Class<T> cls) {
        super(cls);
    }

    @Override
    public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
        TypeDeserializer typeDeserializer)
        throws IOException, JsonProcessingException
    {
        return typeDeserializer.deserializeTypedFromArray(jp, ctxt);
    }
}

/*
/**********************************************************
/* Actual deserializers: efficient String[], char[] deserializers
/**********************************************************
*/

@JacksonStdImpl
final static class StringDeser
    extends ArrayDeser<String[]>
{
    public StringDeser() { super(String[].class); }

    @Override
    public String[] deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        // Ok: must point to START_ARRAY (or equivalent)
        if (!jp.isExpectedStartArrayToken()) {
            throw ctxt.mappingException(_valueClass);
```

```
        }
        final ObjectBuffer buffer = ctxt.leaseObjectBuffer();
        Object[] chunk = buffer.resetAndStart();
        int ix = 0;
        JsonToken t;

        while ((t = jp.nextToken()) != JsonToken.END_ARRAY) {
            // Ok: no need to convert Strings, but must recognize nulls
            String value = (t == JsonToken.VALUE_NULL) ? null : jp.getText();
            if (ix >= chunk.length) {
                chunk = buffer.appendCompletedChunk(chunk);
                ix = 0;
            }
            chunk[ix++] = value;
        }
        String[] result = buffer.completeAndClearBuffer(chunk, ix, String.class);
        ctxt.returnObjectBuffer(buffer);
        return result;
    }
}

@JacksonStdImpl
final static class CharDeser
    extends ArrayDeser<char[]>
{
    public CharDeser() { super(char[].class); }

    @Override
    public char[] deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        /* Won't take arrays, must get a String (could also
         * convert other tokens to Strings... but let's not bother
         * yet, doesn't seem to make sense)
         */
        JsonToken t = jp.getCurrentToken();
        if (t == JsonToken.VALUE_STRING) {
            // note: can NOT return shared internal buffer, must copy:
            char[] buffer = jp.getTextCharacters();
            int offset = jp.getTextOffset();
            int len = jp.getTextLength();

            char[] result = new char[len];
            System.arraycopy(buffer, offset, result, 0, len);
            return result;
        }
        if (jp.isExpectedStartArrayToken()) {
            // Let's actually build as a String, then get chars
```

```
            StringBuilder sb = new StringBuilder(64);
            while ((t = jp.nextToken()) != JsonToken.END_ARRAY) {
                if (t != JsonToken.VALUE_STRING) {
                    throw ctxt.mappingException(Character.TYPE);
                }
                String str = jp.getText();
                if (str.length() != 1) {
                    throw JsonMappingException.from(jp, "Can not convert a JSON String of length "+str.length()+" into
a char element of char array");
                }
                sb.append(str.charAt(0));
            }
            return sb.toString().toCharArray();
        }
        // or, maybe an embedded object?
        if (t == JsonToken.VALUE_EMBEDDED_OBJECT) {
            Object ob = jp.getEmbeddedObject();
            if (ob == null) return null;
            if (ob instanceof char[]) {
                return (char[]) ob;
            }
            if (ob instanceof String) {
                return ((String) ob).toCharArray();
            }
            // 04-Feb-2011, tatu: byte[] can be converted; assuming base64 is wanted
            if (ob instanceof byte[]) {
                return Base64Variants.getDefaultVariant().encode((byte[]) ob, false).toCharArray();
            }
            // not recognized, just fall through
        }
        throw ctxt.mappingException(_valueClass);
    }
}

/*
/**********************************************************
/* Actual deserializers: primivate array desers
/**********************************************************
*/

@JacksonStdImpl
final static class BooleanDeser
    extends ArrayDeser<boolean[]>
{
    public BooleanDeser() { super(boolean[].class); }

    @Override
    public boolean[] deserialize(JsonParser jp, DeserializationContext ctxt)
```

```
        throws IOException, JsonProcessingException
    {
        if (!jp.isExpectedStartArrayToken()) {
            throw ctxt.mappingException(_valueClass);
        }
        ArrayBuilders.BooleanBuilder builder = ctxt.getArrayBuilders().getBooleanBuilder();
        boolean[] chunk = builder.resetAndStart();
        int ix = 0;

        while (jp.nextToken() != JsonToken.END_ARRAY) {
            // whether we should allow truncating conversions?
            boolean value = _parseBooleanPrimitive(jp, ctxt);
            if (ix >= chunk.length) {
                chunk = builder.appendCompletedChunk(chunk, ix);
                ix = 0;
            }
            chunk[ix++] = value;
        }
        return builder.completeAndClearBuffer(chunk, ix);
    }
}

/**
 * When dealing with byte arrays we have one more alternative (compared
 * to int/long/shorts): base64 encoded data.
 */
@JacksonStdImpl
final static class ByteDeser
    extends ArrayDeser<byte[]>
{
    public ByteDeser() { super(byte[].class); }

    @Override
    public byte[] deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();

        // Most likely case: base64 encoded String?
        if (t == JsonToken.VALUE_STRING) {
            return jp.getBinaryValue(ctxt.getBase64Variant());
        }
        // 31-Dec-2009, tatu: Also may be hidden as embedded Object
        if (t == JsonToken.VALUE_EMBEDDED_OBJECT) {
            Object ob = jp.getEmbeddedObject();
            if (ob == null) return null;
            if (ob instanceof byte[]) {
                return (byte[]) ob;
```

```
            }
        }
        if (!jp.isExpectedStartArrayToken()) {
            throw ctxt.mappingException(_valueClass);
        }
        ArrayBuilders.ByteBuilder builder = ctxt.getArrayBuilders().getByteBuilder();
        byte[] chunk = builder.resetAndStart();
        int ix = 0;

        while ((t = jp.nextToken()) != JsonToken.END_ARRAY) {
            // whether we should allow truncating conversions?
            byte value;
            if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) {
                // should we catch overflow exceptions?
                value = jp.getByteValue();
            } else {
                // [JACKSON-79]: should probably accept nulls as 'false'
                if (t != JsonToken.VALUE_NULL) {
                    throw ctxt.mappingException(_valueClass.getComponentType());
                }
                value = (byte) 0;
            }
            if (ix >= chunk.length) {
                chunk = builder.appendCompletedChunk(chunk, ix);
                ix = 0;
            }
            chunk[ix++] = value;
        }
        return builder.completeAndClearBuffer(chunk, ix);
    }
}

@JacksonStdImpl
final static class ShortDeser
    extends ArrayDeser<short[]>
{
    public ShortDeser() { super(short[].class); }

    @Override
    public short[] deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        if (!jp.isExpectedStartArrayToken()) {
            throw ctxt.mappingException(_valueClass);
        }
        ArrayBuilders.ShortBuilder builder = ctxt.getArrayBuilders().getShortBuilder();
        short[] chunk = builder.resetAndStart();
        int ix = 0;
```

```
        while (jp.nextToken() != JsonToken.END_ARRAY) {
            short value = _parseShortPrimitive(jp, ctxt);
            if (ix >= chunk.length) {
                chunk = builder.appendCompletedChunk(chunk, ix);
                ix = 0;
            }
            chunk[ix++] = value;
        }
        return builder.completeAndClearBuffer(chunk, ix);
    }
}

@JacksonStdImpl
final static class IntDeser
    extends ArrayDeser<int[]>
{
    public IntDeser() { super(int[].class); }

    @Override
    public int[] deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        if (!jp.isExpectedStartArrayToken()) {
            throw ctxt.mappingException(_valueClass);
        }
        ArrayBuilders.IntBuilder builder = ctxt.getArrayBuilders().getIntBuilder();
        int[] chunk = builder.resetAndStart();
        int ix = 0;

        while (jp.nextToken() != JsonToken.END_ARRAY) {
            // whether we should allow truncating conversions?
            int value = _parseIntPrimitive(jp, ctxt);
            if (ix >= chunk.length) {
                chunk = builder.appendCompletedChunk(chunk, ix);
                ix = 0;
            }
            chunk[ix++] = value;
        }
        return builder.completeAndClearBuffer(chunk, ix);
    }
}

@JacksonStdImpl
final static class LongDeser
    extends ArrayDeser<long[]>
{
    public LongDeser() { super(long[].class); }
```

```
    @Override
    public long[] deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        if (!jp.isExpectedStartArrayToken()) {
            throw ctxt.mappingException(_valueClass);
        }
        ArrayBuilders.LongBuilder builder = ctxt.getArrayBuilders().getLongBuilder();
        long[] chunk = builder.resetAndStart();
        int ix = 0;

        while (jp.nextToken() != JsonToken.END_ARRAY) {
            long value = _parseLongPrimitive(jp, ctxt);
            if (ix >= chunk.length) {
                chunk = builder.appendCompletedChunk(chunk, ix);
                ix = 0;
            }
            chunk[ix++] = value;
        }
        return builder.completeAndClearBuffer(chunk, ix);
    }
}

@JacksonStdImpl
final static class FloatDeser
    extends ArrayDeser<float[]>
{
    public FloatDeser() { super(float[].class); }

    @Override
    public float[] deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        if (!jp.isExpectedStartArrayToken()) {
            throw ctxt.mappingException(_valueClass);
        }
        ArrayBuilders.FloatBuilder builder = ctxt.getArrayBuilders().getFloatBuilder();
        float[] chunk = builder.resetAndStart();
        int ix = 0;

        while (jp.nextToken() != JsonToken.END_ARRAY) {
            // whether we should allow truncating conversions?
            float value = _parseFloatPrimitive(jp, ctxt);
            if (ix >= chunk.length) {
                chunk = builder.appendCompletedChunk(chunk, ix);
                ix = 0;
            }
```

```
            chunk[ix++] = value;
        }
        return builder.completeAndClearBuffer(chunk, ix);
    }
  }

  @JacksonStdImpl
  final static class DoubleDeser
    extends ArrayDeser<double[]>
  {
    public DoubleDeser() { super(double[].class); }

    @Override
    public double[] deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        if (!jp.isExpectedStartArrayToken()) {
            throw ctxt.mappingException(_valueClass);
        }
        ArrayBuilders.DoubleBuilder builder = ctxt.getArrayBuilders().getDoubleBuilder();
        double[] chunk = builder.resetAndStart();
        int ix = 0;

        while (jp.nextToken() != JsonToken.END_ARRAY) {
            double value = _parseDoublePrimitive(jp, ctxt);
            if (ix >= chunk.length) {
                chunk = builder.appendCompletedChunk(chunk, ix);
                ix = 0;
            }
            chunk[ix++] = value;
        }
        return builder.completeAndClearBuffer(chunk, ix);
    }
  }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.util.Date;

import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.map.DeserializationContext;

/**
* Simple deserializer for handling {@link java.util.Date} values.
*<p>
* One way to customize Date formats accepted is to override method
```

```
* {@link DeserializationContext#parseDate} that this basic
* deserializer calls.
*/
public class DateDeserializer
  extends StdScalarDeserializer<Date>
{
  public DateDeserializer() { super(Date.class); }

  @Override
  public java.util.Date deserialize(JsonParser jp, DeserializationContext ctxt)
     throws IOException, JsonProcessingException
  {
     return _parseDate(jp, ctxt);
  }
}
package org.codehaus.jackson.map.deser;

import java.util.*;

import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.type.*;
import org.codehaus.jackson.type.JavaType;

/**
* Helper class used to contain simple/well-known deserializers for core JDK types
*/
class StdDeserializers
{
  final HashMap<JavaType, JsonDeserializer<Object>> _deserializers = new HashMap<JavaType,
JsonDeserializer<Object>>();

  private StdDeserializers()
  {
     // First, add the fall-back "untyped" deserializer:
     add(new UntypedObjectDeserializer());

     // Then String and String-like converters:
     StdDeserializer<?> strDeser = new StdDeserializer.StringDeserializer();
     add(strDeser, String.class);
     add(strDeser, CharSequence.class);
     add(new StdDeserializer.ClassDeserializer());

     // Then primitive-wrappers (simple):
     add(new StdDeserializer.BooleanDeserializer(Boolean.class, null));
     add(new StdDeserializer.ByteDeserializer(Byte.class, null));
     add(new StdDeserializer.ShortDeserializer(Short.class, null));
     add(new StdDeserializer.CharacterDeserializer(Character.class, null));
     add(new StdDeserializer.IntegerDeserializer(Integer.class, null));
```

```
        add(new StdDeserializer.LongDeserializer(Long.class, null));
        add(new StdDeserializer.FloatDeserializer(Float.class, null));
        add(new StdDeserializer.DoubleDeserializer(Double.class, null));

        /* And actual primitives: difference is the way nulls are to be
         * handled...
         */
        add(new StdDeserializer.BooleanDeserializer(Boolean.TYPE, Boolean.FALSE));
        add(new StdDeserializer.ByteDeserializer(Byte.TYPE, Byte.valueOf((byte)(0))));
        add(new StdDeserializer.ShortDeserializer(Short.TYPE, Short.valueOf((short)0)));
        add(new StdDeserializer.CharacterDeserializer(Character.TYPE, Character.valueOf('\0')));
        add(new StdDeserializer.IntegerDeserializer(Integer.TYPE, Integer.valueOf(0)));
        add(new StdDeserializer.LongDeserializer(Long.TYPE, Long.valueOf(0L)));
        add(new StdDeserializer.FloatDeserializer(Float.TYPE, Float.valueOf(0.0f)));
        add(new StdDeserializer.DoubleDeserializer(Double.TYPE, Double.valueOf(0.0)));

        // and related
        add(new StdDeserializer.NumberDeserializer());
        add(new StdDeserializer.BigDecimalDeserializer());
        add(new StdDeserializer.BigIntegerDeserializer());

        add(new DateDeserializer());
        add(new StdDeserializer.SqlDateDeserializer());
        add(new TimestampDeserializer());
        add(new StdDeserializer.CalendarDeserializer());
        /* 24-Jan-2010, tatu: When including type information, we may
         *    know that we specifically need GregorianCalendar...
         */
        add(new StdDeserializer.CalendarDeserializer(GregorianCalendar.class),
                GregorianCalendar.class);

        // From-string deserializers:
        for (StdDeserializer<?> deser : FromStringDeserializer.all()) {
            add(deser);
        }

        // And finally some odds and ends

        // to deserialize Throwable, need stack trace elements:
        add(new StdDeserializer.StackTraceElementDeserializer());

        // Plus TokenBuffer is a core type since 1.5
        add(new StdDeserializer.TokenBufferDeserializer());
        // [JACKSON-283] need to support atomic types, too
        add(new StdDeserializer.AtomicBooleanDeserializer());
    }

    /**
```

```
     * Public accessor to deserializers for core types.
     */
    public static HashMap<JavaType, JsonDeserializer<Object>> constructAll()
    {
        return new StdDeserializers()._deserializers;
    }

    private void add(StdDeserializer<?> stdDeser)
    {
        add(stdDeser, stdDeser.getValueClass());
    }

    private void add(StdDeserializer<?> stdDeser, Class<?> valueClass)
    {
        // must do some unfortunate casting here...
        @SuppressWarnings("unchecked")
        JsonDeserializer<Object> deser = (JsonDeserializer<Object>) stdDeser;
        _deserializers.put(TypeFactory.type(valueClass), deser);
    }
}
package org.codehaus.jackson.map.deser;

import java.lang.reflect.Constructor;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicReference;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.deser.impl.StringCollectionDeserializer;
import org.codehaus.jackson.map.ext.OptionalHandlerFactory;
import org.codehaus.jackson.map.introspect.*;
import org.codehaus.jackson.map.jsontype.NamedType;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.map.type.*;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.type.JavaType;

/**
* Abstract factory base class that can provide deserializers for standard
* JDK classes, including collection classes and simple heuristics for
* "upcasting" commmon collection interface types
* (such as {@link java.util.Collection}).
*<p>
* Since all simple deserializers are eagerly instantiated, and there is
* no additional introspection or customizability of these types,
* this factory is stateless.
*/
```

```
public abstract class BasicDeserializerFactory
  extends DeserializerFactory
{
  // // Can cache some types

  final static JavaType TYPE_STRING = TypeFactory.type(String.class);

  /**
   * We will pre-create serializers for common non-structured
   * (that is things other than Collection, Map or array)
   * types. These need not go through factory.
   */
  final static HashMap<JavaType, JsonDeserializer<Object>> _simpleDeserializers =
StdDeserializers.constructAll();

  /* We do some defaulting for abstract Map classes and
   * interfaces, to avoid having to use exact types or annotations in
   * cases where the most common concrete Maps will do.
   */
  @SuppressWarnings("unchecked")
  final static HashMap<String, Class<? extends Map>> _mapFallbacks =
     new HashMap<String, Class<? extends Map>>();
  static {
     _mapFallbacks.put(Map.class.getName(), LinkedHashMap.class);
     _mapFallbacks.put(ConcurrentMap.class.getName(), ConcurrentHashMap.class);
     _mapFallbacks.put(SortedMap.class.getName(), TreeMap.class);

     /* 11-Jan-2009, tatu: Let's see if we can still add support for
      *    JDK 1.6 interfaces, even if we run on 1.5. Just need to be
      *    more careful with typos, since compiler won't notice any
      *    problems...
      */
     _mapFallbacks.put("java.util.NavigableMap", TreeMap.class);
     try {
        Class<?> key = Class.forName("java.util.ConcurrentNavigableMap");
        Class<?> value = Class.forName("java.util.ConcurrentSkipListMap");
        @SuppressWarnings("unchecked")
           Class<? extends Map> mapValue = (Class<? extends Map>) value;
        _mapFallbacks.put(key.getName(), mapValue);
     } catch (ClassNotFoundException cnfe) { // occurs on 1.5
     }
  }

  /* We do some defaulting for abstract Collection classes and
   * interfaces, to avoid having to use exact types or annotations in
   * cases where the most common concrete Collection will do.
   */
  @SuppressWarnings("unchecked")
```

```
  final static HashMap<String, Class<? extends Collection>> _collectionFallbacks =
     new HashMap<String, Class<? extends Collection>>();
  static {
     _collectionFallbacks.put(Collection.class.getName(), ArrayList.class);
     _collectionFallbacks.put(List.class.getName(), ArrayList.class);
     _collectionFallbacks.put(Set.class.getName(), HashSet.class);
     _collectionFallbacks.put(SortedSet.class.getName(), TreeSet.class);
     _collectionFallbacks.put(Queue.class.getName(), LinkedList.class);

     /* 11-Jan-2009, tatu: Let's see if we can still add support for
      *    JDK 1.6 interfaces, even if we run on 1.5. Just need to be
      *    more careful with typos, since compiler won't notice any
      *    problems...
      */
     _collectionFallbacks.put("java.util.Deque", LinkedList.class);
     _collectionFallbacks.put("java.util.NavigableSet", TreeSet.class);
  }

  /**
   * And finally, we have special array deserializers for primitive
   * array types
   */
  protected final static HashMap<JavaType,JsonDeserializer<Object>> _arrayDeserializers =
ArrayDeserializers.getAll();

  /**
   * To support external/optional deserializers, we'll use this helper class
   * (as per [JACKSON-386])
   */
  protected OptionalHandlerFactory optionalHandlers = OptionalHandlerFactory.instance;

  /*
  /**********************************************************
  /* Life cycle
  /**********************************************************
   */

  protected BasicDeserializerFactory() { }

  // can't be implemented quite here
  @Override
  public abstract DeserializerFactory withConfig(DeserializerFactory.Config config);

  /*
  /**********************************************************
  /* Methods for sub-classes to override to provide
  /* custom deserializers (since 1.7)
  /**********************************************************
```

```
     */

    protected abstract JsonDeserializer<?> _findCustomArrayDeserializer(ArrayType type, DeserializationConfig
config,
            DeserializerProvider p, BeanProperty property,
            TypeDeserializer elementTypeDeser, JsonDeserializer<?> elementDeser)
        throws JsonMappingException;

    protected abstract JsonDeserializer<?> _findCustomCollectionDeserializer(CollectionType type,
DeserializationConfig config,
            DeserializerProvider p, BasicBeanDescription beanDesc, BeanProperty property,
            TypeDeserializer elementTypeDeser, JsonDeserializer<?> elementDeser)
        throws JsonMappingException;

    protected abstract JsonDeserializer<?> _findCustomEnumDeserializer(Class<?> type, DeserializationConfig
config,
            BasicBeanDescription beanDesc, BeanProperty property)
        throws JsonMappingException;

    protected abstract JsonDeserializer<?> _findCustomMapDeserializer(MapType type, DeserializationConfig
config,
            DeserializerProvider p, BasicBeanDescription beanDesc, BeanProperty property,
            KeyDeserializer keyDeser,
            TypeDeserializer elementTypeDeser, JsonDeserializer<?> elementDeser)
        throws JsonMappingException;

    protected abstract JsonDeserializer<?> _findCustomTreeNodeDeserializer(Class<? extends JsonNode> type,
            DeserializationConfig config, BeanProperty property)
        throws JsonMappingException;

    /*
    /**********************************************************
    /* JsonDeserializerFactory impl
    /**********************************************************
     */

    @Override
    public JsonDeserializer<?> createArrayDeserializer(DeserializationConfig config, DeserializerProvider p,
            ArrayType type, BeanProperty property)
        throws JsonMappingException
    {
        JavaType elemType = type.getContentType();

        // Very first thing: is deserializer hard-coded for elements?
        JsonDeserializer<Object> contentDeser = elemType.getValueHandler();
        if (contentDeser == null) {
            // Maybe special array type, such as "primitive" arrays (int[] etc)
            JsonDeserializer<?> deser = _arrayDeserializers.get(elemType);
```

```
            if (deser != null) {
                /* 23-Nov-2010, tatu: Although not commonly needed, ability to override
                 *   deserializers for all types (including primitive arrays) is useful
                 *   so let's allow this
                 */
                JsonDeserializer<?> custom = _findCustomArrayDeserializer(type, config, p, property, null, null);
                if (custom != null) {
                    return custom;
                }
                return deser;
            }
            // If not, generic one:
            if (elemType.isPrimitive()) { // sanity check
                throw new IllegalArgumentException("Internal error: primitive type ("+type+") passed, no array
deserializer found");
            }
        }
        // Then optional type info (1.5): if type has been resolved, we may already know type deserializer:
        TypeDeserializer elemTypeDeser = elemType.getTypeHandler();
        // but if not, may still be possible to find:
        if (elemTypeDeser == null) {
            elemTypeDeser = findTypeDeserializer(config, elemType, property);
        }
        // 23-Nov-2010, tatu: Custom array deserializer?
        JsonDeserializer<?> custom = _findCustomArrayDeserializer(type, config, p, property, elemTypeDeser,
contentDeser);
        if (custom != null) {
            return custom;
        }

        if (contentDeser == null) {
            // 'null' -> arrays have no referring fields
            contentDeser = p.findValueDeserializer(config, elemType, property);
        }
        return new ArrayDeserializer(type, contentDeser, elemTypeDeser);
    }

    @Override
    public JsonDeserializer<?> createCollectionDeserializer(DeserializationConfig config, DeserializerProvider p,
            CollectionType type, BeanProperty property)
        throws JsonMappingException
    {
        Class<?> collectionClass = type.getRawClass();
        BasicBeanDescription beanDesc = config.introspectClassAnnotations(collectionClass);
        // Explicit deserializer to use? (@JsonDeserialize.using)
        JsonDeserializer<Object> deser = findDeserializerFromAnnotation(config, beanDesc.getClassInfo(), property);
        if (deser != null) {
            return deser;
```

```
        }
        // If not, any type modifiers? (@JsonDeserialize.as)
        type = modifyTypeByAnnotation(config, beanDesc.getClassInfo(), type, null);

        JavaType contentType = type.getContentType();
        // Very first thing: is deserializer hard-coded for elements?
        JsonDeserializer<Object> contentDeser = contentType.getValueHandler();

        // Then optional type info (1.5): if type has been resolved, we may already know type deserializer:
        TypeDeserializer contentTypeDeser = contentType.getTypeHandler();
        // but if not, may still be possible to find:
        if (contentTypeDeser == null) {
           contentTypeDeser = findTypeDeserializer(config, contentType, property);
        }

        // 23-Nov-2010, tatu: Custom deserializer?
        JsonDeserializer<?> custom = _findCustomCollectionDeserializer(type, config, p, beanDesc, property,
              contentTypeDeser, contentDeser);
        if (custom != null) {
           return custom;
        }

        if (contentDeser == null) { // not defined by annotation
           // One special type: EnumSet:
           if (EnumSet.class.isAssignableFrom(collectionClass)) {
              return new EnumSetDeserializer(constructEnumResolver(contentType.getRawClass(), config));
           }
           // But otherwise we can just use a generic value deserializer:
           // 'null' -> collections have no referring fields
           contentDeser = p.findValueDeserializer(config, contentType, property);
        }

        /* One twist: if we are being asked to instantiate an interface or
         * abstract Collection, we need to either find something that implements
         * the thing, or give up.
         *
         * Note that we do NOT try to guess based on secondary interfaces
         * here; that would probably not work correctly since casts would
         * fail later on (as the primary type is not the interface we'd
         * be implementing)
         */
        if (type.isInterface() || type.isAbstract()) {
           @SuppressWarnings("unchecked")
           Class<? extends Collection> fallback = _collectionFallbacks.get(collectionClass.getName());
           if (fallback == null) {
              throw new IllegalArgumentException("Can not find a deserializer for non-concrete Collection type
"+type);
           }
```

```
            collectionClass = fallback;
        }

        boolean fixAccess =
config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS);
        @SuppressWarnings("unchecked")
        Constructor<Collection<Object>> ctor =
ClassUtil.findConstructor((Class<Collection<Object>>)collectionClass, fixAccess);
        // 13-Dec-2010, tatu: Can use more optimal deserializer if content type is String, so:
        if (contentType.getRawClass() == String.class) {
            // no value type deserializer because Strings are one of natural/native types:
            return new StringCollectionDeserializer(type, contentDeser, ctor);
        }
        return new CollectionDeserializer(type, contentDeser, contentTypeDeser, ctor);
    }

    @Override
    public JsonDeserializer<?> createMapDeserializer(DeserializationConfig config, DeserializerProvider p,
            MapType type, BeanProperty property)
        throws JsonMappingException
    {
        Class<?> mapClass = type.getRawClass();

        BasicBeanDescription beanDesc = config.introspectForCreation(type);
        // Explicit deserializer to use? (@JsonDeserialize.using)
        JsonDeserializer<Object> deser = findDeserializerFromAnnotation(config, beanDesc.getClassInfo(), property);
        if (deser != null) {
            return deser;
        }
        // If not, any type modifiers? (@JsonDeserialize.as)
        type = modifyTypeByAnnotation(config, beanDesc.getClassInfo(), type, null);

        JavaType keyType = type.getKeyType();
        JavaType contentType = type.getContentType();

        // First: is there annotation-specified deserializer for values?
        @SuppressWarnings("unchecked")
        JsonDeserializer<Object> contentDeser = (JsonDeserializer<Object>) contentType.getValueHandler();

        // Ok: need a key deserializer (null indicates 'default' here)
        KeyDeserializer keyDes = (KeyDeserializer) keyType.getValueHandler();
        if (keyDes == null) {
            keyDes = (TYPE_STRING.equals(keyType)) ? null : p.findKeyDeserializer(config, keyType, property);
        }
        // Then optional type info (1.5); either attached to type, or resolve separately:
        TypeDeserializer contentTypeDeser = contentType.getTypeHandler();
        // but if not, may still be possible to find:
        if (contentTypeDeser == null) {
```

```
    contentTypeDeser = findTypeDeserializer(config, contentType, property);
}

// 23-Nov-2010, tatu: Custom deserializer?
JsonDeserializer<?> custom = _findCustomMapDeserializer(type, config, p, beanDesc, property,
        keyDes, contentTypeDeser, contentDeser);
if (custom != null) {
    return custom;
}

if (contentDeser == null) { // nope...
    // 'null' -> maps have no referring fields
    contentDeser = p.findValueDeserializer(config, contentType, property);
}
/* Value handling is identical for all,
 * but EnumMap requires special handling for keys
 */
if (EnumMap.class.isAssignableFrom(mapClass)) {
    Class<?> kt = keyType.getRawClass();
    if (kt == null || !kt.isEnum()) {
        throw new IllegalArgumentException("Can not construct EnumMap; generic (key) type not available");
    }
    return new EnumMapDeserializer(constructEnumResolver(kt, config), contentDeser);
}

// Otherwise, generic handler works ok.

/* But there is one more twist: if we are being asked to instantiate
 * an interface or abstract Map, we need to either find something
 * that implements the thing, or give up.
 *
 * Note that we do NOT try to guess based on secondary interfaces
 * here; that would probably not work correctly since casts would
 * fail later on (as the primary type is not the interface we'd
 * be implementing)
 */
if (type.isInterface() || type.isAbstract()) {
    @SuppressWarnings("unchecked")
    Class<? extends Map> fallback = _mapFallbacks.get(mapClass.getName());
    if (fallback == null) {
        throw new IllegalArgumentException("Can not find a deserializer for non-concrete Map type "+type);
    }
    mapClass = fallback;
    type = (MapType) type.forcedNarrowBy(mapClass);
    // But if so, also need to re-check creators...
    beanDesc = config.introspectForCreation(type);
}
```

```
        // [JACKSON-153]: allow use of @JsonCreator
        boolean fixAccess =
    config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS);
        // First, locate the default constructor (if one available)
        @SuppressWarnings("unchecked")
        Constructor<Map<Object,Object>> defaultCtor = (Constructor<Map<Object,Object>>)
    beanDesc.findDefaultConstructor();
        if (defaultCtor != null) {
            if (fixAccess) {
                ClassUtil.checkAndFixAccess(defaultCtor);
            }
        }
        MapDeserializer md = new MapDeserializer(type, defaultCtor, keyDes, contentDeser, contentTypeDeser);
    md.setIgnorableProperties(config.getAnnotationIntrospector().findPropertiesToIgnore(beanDesc.getClassInfo()));
      md.setCreators(findMapCreators(config, beanDesc));
        return md;
      }

      /**
       * Factory method for constructing serializers of {@link Enum} types.
       */
      @Override
      public JsonDeserializer<?> createEnumDeserializer(DeserializationConfig config, DeserializerProvider p,
            JavaType type, BeanProperty property)
        throws JsonMappingException
      {
        /* 18-Feb-2009, tatu: Must first check if we have a class annotation
         *    that should override default deserializer
         */
        BasicBeanDescription beanDesc = config.introspectForCreation(type);
        JsonDeserializer<?> des = findDeserializerFromAnnotation(config, beanDesc.getClassInfo(), property);
        if (des != null) {
            return des;
        }
        Class<?> enumClass = type.getRawClass();
        // 23-Nov-2010, tatu: Custom deserializer?
        JsonDeserializer<?> custom = _findCustomEnumDeserializer(enumClass, config, beanDesc, property);
        if (custom != null) {
            return custom;
        }

        // [JACKSON-193] May have @JsonCreator for static factory method:
        for (AnnotatedMethod factory : beanDesc.getFactoryMethods()) {
            if (config.getAnnotationIntrospector().hasCreatorAnnotation(factory)) {
                int argCount = factory.getParameterCount();
                if (argCount == 1) {
                    Class<?> returnType = factory.getRawType();
                    // usually should be class, but may be just plain Enum<?> (for Enum.valueOf()?)
```

```
                if (returnType.isAssignableFrom(enumClass)) {
                    return EnumDeserializer.deserializerForCreator(config, enumClass, factory);
                }
            }
            throw new IllegalArgumentException("Unsuitable method ("+factory+") decorated with @JsonCreator (for
Enum type "
                    +enumClass.getName()+")");
        }
    }
    return new EnumDeserializer(constructEnumResolver(enumClass, config));
  }

  @Override
  public JsonDeserializer<?> createTreeDeserializer(DeserializationConfig config, DeserializerProvider p,
        JavaType nodeType, BeanProperty property)
    throws JsonMappingException
  {
    @SuppressWarnings("unchecked")
    Class<? extends JsonNode> nodeClass = (Class<? extends JsonNode>) nodeType.getRawClass();
    // 23-Nov-2010, tatu: Custom deserializer?
    JsonDeserializer<?> custom = _findCustomTreeNodeDeserializer(nodeClass, config, property);
    if (custom != null) {
        return custom;
    }
    return JsonNodeDeserializer.getDeserializer(nodeClass);
  }

  @SuppressWarnings("unchecked")
  @Override
  public JsonDeserializer<Object> createBeanDeserializer(DeserializationConfig config, DeserializerProvider p,
        JavaType type, BeanProperty property)
    throws JsonMappingException
  {
    // note: we do NOT check for custom deserializers here; that's for sub-class to do

    JsonDeserializer<Object> deser = _simpleDeserializers.get(type);
    if (deser != null) {
        return deser;
    }
    // [JACKSON-283]: AtomicReference is a rather special type...
    Class<?> cls = type.getRawClass();
    if (AtomicReference.class.isAssignableFrom(cls)) {
        JsonDeserializer<?> d2 = new StdDeserializer.AtomicReferenceDeserializer(type, property);
        return (JsonDeserializer<Object>)d2;
    }
    // [JACKSON-386]: External/optional type handlers are handled somewhat differently
    JsonDeserializer<?> d = optionalHandlers.findDeserializer(type, config, p);
    if (d != null) {
```

```
            return (JsonDeserializer<Object>)d;
        }
        return null;
    }

    @Override
    public TypeDeserializer findTypeDeserializer(DeserializationConfig config, JavaType baseType,
            BeanProperty property)
    {
        Class<?> cls = baseType.getRawClass();
        BasicBeanDescription bean = config.introspectClassAnnotations(cls);
        AnnotatedClass ac = bean.getClassInfo();
        AnnotationIntrospector ai = config.getAnnotationIntrospector();
        TypeResolverBuilder<?> b = ai.findTypeResolver(ac, baseType);

        /* Ok: if there is no explicit type info handler, we may want to
         * use a default. If so, config object knows what to use.
         */
        Collection<NamedType> subtypes = null;
        if (b == null) {
            b = config.getDefaultTyper(baseType);
            if (b == null) {
                return null;
            }
        } else {
            subtypes = config.getSubtypeResolver().collectAndResolveSubtypes(ac, config, ai);
        }
        return b.buildTypeDeserializer(baseType, subtypes, property);
    }

    /*
    /**********************************************************
    /* Extended API
    /**********************************************************
     */

    /**
     * Method called to create a type information deserializer for values of
     * given non-container property, if one is needed.
     * If not needed (no polymorphic handling configured for property), should return null.
     *<p>
     * Note that this method is only called for non-container bean properties,
     * and not for values in container types or root values (or container properties)
     *
     * @param baseType Declared base type of the value to deserializer (actual
     *    deserializer type will be this type or its subtype)
     *
     * @return Type deserializer to use for given base type, if one is needed; null if not.
```

```
     *
     * @since 1.5
     */
    public TypeDeserializer findPropertyTypeDeserializer(DeserializationConfig config, JavaType baseType,
            AnnotatedMember annotated, BeanProperty property)
    {
        AnnotationIntrospector ai = config.getAnnotationIntrospector();
        TypeResolverBuilder<?> b = ai.findPropertyTypeResolver(annotated, baseType);

        // Defaulting: if no annotations on member, check value class
        if (b == null) {
            return findTypeDeserializer(config, baseType, property);
        }
        // but if annotations found, may need to resolve subtypes:
        Collection<NamedType> subtypes = config.getSubtypeResolver().collectAndResolveSubtypes(annotated,
config, ai);
        return b.buildTypeDeserializer(baseType, subtypes, property);
    }

    /**
     * Method called to find and create a type information deserializer for values of
     * given container (list, array, map) property, if one is needed.
     * If not needed (no polymorphic handling configured for property), should return null.
     *<p>
     * Note that this method is only called for container bean properties,
     * and not for values in container types or root values (or non-container properties)
     *
     * @param containerType Type of property; must be a container type
     * @param propertyEntity Field or method that contains container property
     *
     * @since 1.5
     */
    public TypeDeserializer findPropertyContentTypeDeserializer(DeserializationConfig config, JavaType
containerType,
            AnnotatedMember propertyEntity, BeanProperty property)
    {
        AnnotationIntrospector ai = config.getAnnotationIntrospector();
        TypeResolverBuilder<?> b = ai.findPropertyContentTypeResolver(propertyEntity, containerType);
        JavaType contentType = containerType.getContentType();
        // Defaulting: if no annotations on member, check class
        if (b == null) {
            return findTypeDeserializer(config, contentType, property);
        }
        // but if annotations found, may need to resolve subtypes:
        Collection<NamedType> subtypes = config.getSubtypeResolver().collectAndResolveSubtypes(propertyEntity,
config, ai);
        return b.buildTypeDeserializer(contentType, subtypes, property);
    }
```

```
    /*
    /**********************************************************
    /* Helper methods, value/content/key type introspection
    /**********************************************************
     */

    /**
     * Helper method called to check if a class or method
     * has annotation that tells which class to use for deserialization.
     * Returns null if no such annotation found.
     */
    protected JsonDeserializer<Object> findDeserializerFromAnnotation(DeserializationConfig config, Annotated a,
            BeanProperty property)
    {
        Object deserDef = config.getAnnotationIntrospector().findDeserializer(a, property);
        if (deserDef != null) {
            return _constructDeserializer(config, deserDef);
        }
        return null;
    }

    @SuppressWarnings("unchecked")
    JsonDeserializer<Object> _constructDeserializer(DeserializationConfig config, Object deserDef)
    {
        if (deserDef instanceof JsonDeserializer) {
            return (JsonDeserializer<Object>) deserDef;
        }
        /* Alas, there's no way to force return type of "either class
         * X or Y" -- need to throw an exception after the fact
         */
        if (!(deserDef instanceof Class)) {
            throw new IllegalStateException("AnnotationIntrospector returned deserializer definition of type
"+deserDef.getClass().getName()+"; expected type JsonDeserializer or Class<JsonDeserializer> instead");
        }
        Class<?> cls = (Class<?>) deserDef;
        if (!JsonDeserializer.class.isAssignableFrom(cls)) {
            throw new IllegalStateException("AnnotationIntrospector returned Class "+cls.getName()+"; expected
Class<JsonDeserializer>");
        }
        return (JsonDeserializer<Object>) ClassUtil.createInstance(cls,
config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS));
    }

    /**
     * Method called to see if given method has annotations that indicate
     * a more specific type than what the argument specifies.
     * If annotations are present, they must specify compatible Class;
```

```
     * instance of which can be assigned using the method. This means
     * that the Class has to be raw class of type, or its sub-class
     * (or, implementing class if original Class instance is an interface).
     *
     * @param a Method or field that the type is associated with
     * @param type Type derived from the setter argument
     * @param propName Name of property that refers to type, if any; null
     *   if no property information available (when modify type declaration
     *   of a class, for example)
     *
     * @return Original type if no annotations are present; or a more
     *   specific type derived from it if type annotation(s) was found
     *
     * @throws JsonMappingException if invalid annotation is found
     */
    @SuppressWarnings("unchecked")
    protected <T extends JavaType> T modifyTypeByAnnotation(DeserializationConfig config,
            Annotated a, T type, String propName)
        throws JsonMappingException
    {
        // first: let's check class for the instance itself:
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
        Class<?> subclass = intr.findDeserializationType(a, type, propName);
        if (subclass != null) {
            try {
                type = (T) type.narrowBy(subclass);
            } catch (IllegalArgumentException iae) {
                throw new JsonMappingException("Failed to narrow type "+type+" with concrete-type annotation (value
"+subclass.getName()+"), method '"+a.getName()+"': "+iae.getMessage(), null, iae);
            }
        }

        // then key class
        if (type.isContainerType()) {
            Class<?> keyClass = intr.findDeserializationKeyType(a, type.getKeyType(), propName);
            if (keyClass != null) {
                // illegal to use on non-Maps
                if (!(type instanceof MapType)) {
                    throw new JsonMappingException("Illegal key-type annotation: type "+type+" is not a Map type");
                }
                try {
                    type = (T) ((MapType) type).narrowKey(keyClass);
                } catch (IllegalArgumentException iae) {
                    throw new JsonMappingException("Failed to narrow key type "+type+" with key-type annotation
("+keyClass.getName()+"): "+iae.getMessage(), null, iae);
                }
            }
```

```
        // and finally content class; only applicable to structured types
        Class<?> cc = intr.findDeserializationContentType(a, type.getContentType(), propName);
        if (cc != null) {
            try {
                type = (T) type.narrowContentsBy(cc);
            } catch (IllegalArgumentException iae) {
                throw new JsonMappingException("Failed to narrow content type "+type+" with content-type
annotation ("+cc.getName()+"): "+iae.getMessage(), null, iae);
            }
        }
    }
    return type;
  }

  /**
   * Helper method used to resolve method return types and field
   * types. The main trick here is that the containing bean may
   * have type variable binding information (when deserializing
   * using generic type passed as type reference), which is
   * needed in some cases.
   *<p>
   * Starting with version 1.3, this method will also resolve instances
   * of key and content deserializers if defined by annotations.
   */
  protected JavaType resolveType(DeserializationConfig config,
        BasicBeanDescription beanDesc, JavaType type, AnnotatedMember member,
        BeanProperty property)
  {
    // [JACKSON-154]: Also need to handle keyUsing, contentUsing
    if (type.isContainerType()) {
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
        boolean canForceAccess =
config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS);
        JavaType keyType = type.getKeyType();
        if (keyType != null) {
            Class<? extends KeyDeserializer> kdClass = intr.findKeyDeserializer(member);
            if (kdClass != null && kdClass != KeyDeserializer.None.class) {
                KeyDeserializer kd = ClassUtil.createInstance(kdClass, canForceAccess);
                keyType.setValueHandler(kd);
            }
        }
        // and all container types have content types...
        Class<? extends JsonDeserializer<?>> cdClass = intr.findContentDeserializer(member);
        if (cdClass != null && cdClass != JsonDeserializer.None.class) {
            JsonDeserializer<?> cd = ClassUtil.createInstance(cdClass, canForceAccess);
            type.getContentType().setValueHandler(cd);
        }
        /* 04-Feb-2010, tatu: Need to figure out JAXB annotations that indicate type
```

```
         *    information to use for polymorphic members; and specifically types for
         *    collection values (contents).
         *    ... but only applies to members (fields, methods), not classes
         */
        if (member instanceof AnnotatedMember) {
         TypeDeserializer contentTypeDeser = findPropertyContentTypeDeserializer(config, type,
              (AnnotatedMember) member, property);
         if (contentTypeDeser != null) {
            type = type.withContentTypeHandler(contentTypeDeser);
         }
        }
    }
 TypeDeserializer valueTypeDeser;

    if (member instanceof AnnotatedMember) { // JAXB allows per-property annotations
       valueTypeDeser = findPropertyTypeDeserializer(config, type, (AnnotatedMember) member, property);
    } else { // classes just have Jackson annotations
       // probably only occurs if 'property' is null anyway
       valueTypeDeser = findTypeDeserializer(config, type, null);
    }
 if (valueTypeDeser != null) {
       type = type.withTypeHandler(valueTypeDeser);
 }
    return type;
}

protected EnumResolver<?> constructEnumResolver(Class<?> enumClass, DeserializationConfig config)
{
    // [JACKSON-212]: may need to use Enum.toString()
    if (config.isEnabled(DeserializationConfig.Feature.READ_ENUMS_USING_TO_STRING)) {
       return EnumResolver.constructUnsafeUsingToString(enumClass);
    }
    return EnumResolver.constructUnsafe(enumClass, config.getAnnotationIntrospector());
}

/*
/**********************************************************
/* Helper methods, dealing with Creators
/**********************************************************
 */

/**
 * Method used to find non-default constructors and factory
 * methods that are marked to be used as Creators for a Map type.
 */
protected CreatorContainer findMapCreators(DeserializationConfig config, BasicBeanDescription beanDesc)
    throws JsonMappingException
{
```

```
        Class<?> mapClass = beanDesc.getBeanClass();
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
        boolean fixAccess =
    config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS);
        CreatorContainer creators =  new CreatorContainer(mapClass, fixAccess);
        // First, let's find if we have a constructor creator:
        for (AnnotatedConstructor ctor : beanDesc.getConstructors()) {
            int argCount = ctor.getParameterCount();
            if (argCount < 1 || !intr.hasCreatorAnnotation(ctor)) { // default ctor, or not marked with JsonCreator, skip
                continue;
            }
            // For Map types property name is not optional for ctor params
            SettableBeanProperty[] properties = new SettableBeanProperty[argCount];
            int nameCount = 0;
            for (int i = 0; i < argCount; ++i) {
                AnnotatedParameter param = ctor.getParameter(i);
                String name = (param == null) ? null : intr.findPropertyNameForParam(param);
                // At this point, name annotation is NOT optional
                if (name == null || name.length() == 0) {
                    throw new IllegalArgumentException("Parameter #"+i+" of constructor "+ctor+" has no property name
    annotation: must have for @JsonCreator for a Map type");
                }
                ++nameCount;
                properties[i] = constructCreatorProperty(config, beanDesc, name, i, param);
            }
            creators.addPropertyConstructor(ctor, properties);
        }

        // And then if there's a factory creator
        for (AnnotatedMethod factory : beanDesc.getFactoryMethods()) {
            int argCount = factory.getParameterCount();
            if (argCount < 1 || !intr.hasCreatorAnnotation(factory)) { // no args, or not marked with JsonCreator, skip
                continue;
            }
            // Property name is not optional for factory method params
            SettableBeanProperty[] properties = new SettableBeanProperty[argCount];
            int nameCount = 0;
            for (int i = 0; i < argCount; ++i) {
                AnnotatedParameter param = factory.getParameter(i);
                String name = (param == null) ? null : intr.findPropertyNameForParam(param);
                // At this point, name annotation is NOT optional
                if (name == null || name.length() == 0) {
                    throw new IllegalArgumentException("Parameter #"+i+" of factory method "+factory+" has no property
    name annotation: must have for @JsonCreator for a Map type");
                }
                ++nameCount;
                properties[i] = constructCreatorProperty(config, beanDesc, name, i, param);
            }
```

```
            creators.addPropertyFactory(factory, properties);
        }
        return creators;
    }

    /**
     * Method that will construct a property object that represents
     * a logical property passed via Creator (constructor or static
     * factory method)
     */
    protected SettableBeanProperty constructCreatorProperty(DeserializationConfig config,
            BasicBeanDescription beanDesc, String name, int index,
            AnnotatedParameter param)
        throws JsonMappingException
    {
        JavaType t0 = TypeFactory.type(param.getParameterType(), beanDesc.bindingsForBeanType());
        BeanProperty.Std property = new BeanProperty.Std(name, t0, beanDesc.getClassAnnotations(), param);
        JavaType type = resolveType(config, beanDesc, t0, param, property);
        if (type != t0) {
            property = property.withType(type);
        }
        // Is there an annotation that specifies exact deserializer?
        JsonDeserializer<Object> deser = findDeserializerFromAnnotation(config, param, property);
        // If yes, we are mostly done:
        type = modifyTypeByAnnotation(config, param, type, name);
        TypeDeserializer typeDeser = findTypeDeserializer(config, type, property);
        SettableBeanProperty prop = new SettableBeanProperty.CreatorProperty(name, type, typeDeser,
                beanDesc.getClassAnnotations(), param, index);
        if (deser != null) {
            prop.setValueDeserializer(deser);
        }
        return prop;
    }
}
package org.codehaus.jackson.map.deser;

import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.map.DeserializationContext;

/**
* Simple container used for temporarily buffering a set of
* <code>PropertyValue</code>s.
* Using during construction of beans (and Maps) that use Creators,
* and hence need buffering before instance (that will have properties
* to assign values to) is constructed.
*/
public final class PropertyValueBuffer
{
```

```
final JsonParser _parser;
final DeserializationContext _context;

/**
 * Buffer used for storing creator parameters for constructing
 * instance
 */
final Object[] _creatorParameters;

/**
 * Number of creator parameters we are still missing.
 *<p>
 * NOTE: assumes there are no duplicates, for now.
 */
private int _paramsNeeded;

/**
 * If we get non-creator parameters before or between
 * creator parameters, those need to be buffered. Buffer
 * is just a simple linked list
 */
private PropertyValue _buffered;

public PropertyValueBuffer(JsonParser jp, DeserializationContext ctxt,
                           int paramCount)
{
   _parser = jp;
   _context = ctxt;
   _paramsNeeded = paramCount;
   _creatorParameters = new Object[paramCount];
}

/**
 * @param defaults If any of parameters requires nulls to be replaced with a non-null
 *    object (usually primitive types), this is a non-null array that has such replacement
 *    values (and nulls for cases where nulls are ok)
 */
protected final Object[] getParameters(Object[] defaults)
{
   if (defaults != null) {
      for (int i = 0, len = _creatorParameters.length; i < len; ++i) {
         if (_creatorParameters[i] == null) {
            Object value = defaults[i];
            if (value != null) {
               _creatorParameters[i] = value;
            }
         }
      }
```

```
        }
        return _creatorParameters;
    }

    protected PropertyValue buffered() { return _buffered; }

    /**
     * @return True if we have received all creator parameters
     */
    public boolean assignParameter(int index, Object value) {
        _creatorParameters[index] = value;
        return --_paramsNeeded <= 0;
    }

    public void bufferProperty(SettableBeanProperty prop, Object value) {
        _buffered = new PropertyValue.Regular(_buffered, value, prop);
    }

    public void bufferAnyProperty(SettableAnyProperty prop, String propName, Object value) {
        _buffered = new PropertyValue.Any(_buffered, value, prop, propName);
    }

    public void bufferMapProperty(Object key, Object value) {
        _buffered = new PropertyValue.Map(_buffered, value, key);
    }
}
package org.codehaus.jackson.map.deser.impl;

import java.util.*;

import org.codehaus.jackson.map.deser.SettableBeanProperty;

/**
* Helper class used for storing mapping from property name to
* {@link SettableBeanProperty} instances.
*<p>
* Note that this class is used instead of generic {@link java.util.HashMap}
* is performance: although default implementation is very good for generic
* use cases, it can still be streamlined a bit for specific use case
* we have.
*
* @since 1.7
*/
public final class BeanPropertyMap
{
    private final Bucket[] _buckets;

    private final int _hashMask;
```

```
private final int _size;

public BeanPropertyMap(Collection<SettableBeanProperty> properties)
{
   _size = properties.size();
   int bucketCount = findSize(_size);
   _hashMask = bucketCount-1;
   Bucket[] buckets = new Bucket[bucketCount];
   for (SettableBeanProperty property : properties) {
      String key = property.getName();
      int index = key.hashCode() & _hashMask;
      buckets[index] = new Bucket(buckets[index], key, property);
   }
   _buckets = buckets;
}

public void assignIndexes()
{
   // order is arbitrary, but stable:
   int index = 0;
   for (Bucket bucket : _buckets) {
      while (bucket != null) {
         bucket.value.assignIndex(index++);
         bucket = bucket.next;
      }
   }
}

private final static int findSize(int size)
{
   // For small enough results (32 or less), we'll require <= 50% fill rate; otherwise 80%
   int needed = (size <= 32) ? (size + size) : (size + (size >> 2));
   int result = 2;
   while (result < needed) {
      result += result;
   }
   return result;
}

/*
/**********************************************************
/* Public API
/**********************************************************
 */

public int size() { return _size; }
```

```
/**
 * Accessor for traversing over all contained properties.
 */
public Iterator<SettableBeanProperty> allProperties() {
   return new IteratorImpl(_buckets);
}

public SettableBeanProperty find(String key)
{
   int index = key.hashCode() & _hashMask;
   Bucket bucket = _buckets[index];
   // Let's unroll first lookup since that is null or match in 90+% cases
   if (bucket == null) {
      return null;
   }
   // Primarily we do just identity comparison as keys should be interned
   if (bucket.key == key) {
      return bucket.value;
   }
   while ((bucket = bucket.next) != null) {
      if (bucket.key == key) {
         return bucket.value;
      }
   }
   // Do we need fallback for non-interned Strings?
   return _findWithEquals(key, index);
}

/**
 * Specialized method that can be used to replace an existing entry
 * (note: entry MUST exist; otherwise exception is thrown) with
 * specified replacement.
 */
public void replace(SettableBeanProperty property)
{
   String name = property.getName();
   int index = name.hashCode() & (_buckets.length-1);

   /* This is bit tricky just because buckets themselves
    * are immutable, so we need to recreate the chain. Fine.
    */
   Bucket tail = null;
   boolean found = false;


   for (Bucket bucket = _buckets[index]; bucket != null; bucket = bucket.next) {
      // match to remove?
      if (!found && bucket.key.equals(name)) {
```

```
            found = true;
        } else {
            tail = new Bucket(tail, bucket.key, bucket.value);
        }
    }
    // Not finding specified entry is error, so:
    if (!found) {
        throw new NoSuchElementException("No entry '"+property+"' found, can't replace");
    }
    // So let's attach replacement in front:
    _buckets[index] = new Bucket(tail, name, property);
}

/*
/**********************************************************
/* Helper methods
/**********************************************************
 */

private SettableBeanProperty _findWithEquals(String key, int index)
{
    Bucket bucket = _buckets[index];
    while (bucket != null) {
        if (key.equals(bucket.key)) {
            return bucket.value;
        }
        bucket = bucket.next;
    }
    return null;
}

/*
/**********************************************************
/* Helper beans
/**********************************************************
 */

private final static class Bucket
{
    public final Bucket next;
    public final String key;
    public final SettableBeanProperty value;

    public Bucket(Bucket next, String key, SettableBeanProperty value)
    {
        this.next = next;
        this.key = key;
        this.value = value;
```

```
    }
}

private final static class IteratorImpl implements Iterator<SettableBeanProperty>
{
    /**
     * Buckets of the map
     */
    private final Bucket[] _buckets;

    /**
     * Bucket that contains next value to return (if any); null if nothing more to iterate
     */
    private Bucket _currentBucket;

    /**
     * Index of the next bucket in bucket array to check.
     */
    private int _nextBucketIndex;

    public IteratorImpl(Bucket[] buckets) {
        _buckets = buckets;
        // need to initialize to point to first entry...
        int i = 0;
        for (int len = _buckets.length; i < len; ) {
            Bucket b = _buckets[i++];
            if (b != null) {
                _currentBucket = b;
                break;
            }
        }
        _nextBucketIndex = i;
    }

    @Override
    public boolean hasNext() {
        return _currentBucket != null;
    }

    @Override
    public SettableBeanProperty next()
    {
        Bucket curr = _currentBucket;
        if (curr == null) { // sanity check
            throw new NoSuchElementException();
        }
        // need to advance, too
        Bucket b = curr.next;
```

```
            while (b == null && _nextBucketIndex < _buckets.length) {
                b = _buckets[_nextBucketIndex++];
            }
            _currentBucket = b;
            return curr.value;
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
package org.codehaus.jackson.map.deser.impl;

import java.io.IOException;
import java.lang.reflect.Constructor;
import java.util.Collection;

import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.TypeDeserializer;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.deser.ContainerDeserializer;
import org.codehaus.jackson.type.JavaType;

@JacksonStdImpl
public final class StringCollectionDeserializer
    extends ContainerDeserializer<Collection<String>>
{
    // // Configuration

    protected final JavaType _collectionType;

    /**
     * Value deserializer; needed even if it is the standard String
     * deserializer
     */
    protected final JsonDeserializer<String> _valueDeserializer;

    /**
     * Flag that indicates whether value deserializer is the standard
     * Jackson-provided one; if it is, we can use more efficient
     * handling.
     */
```

```
protected final boolean _isDefaultDeserializer;

/**
 * We will use the default constructor of the collection class for
 * instantiating result.
 */
final Constructor<Collection<String>> _defaultCtor;

@SuppressWarnings("unchecked")
public StringCollectionDeserializer(JavaType collectionType, JsonDeserializer<?> valueDeser,
      Constructor<?> ctor)
{
   super(collectionType.getRawClass());
   _collectionType = collectionType;
   _valueDeserializer = (JsonDeserializer<String>) valueDeser;
   _defaultCtor = (Constructor<Collection<String>>) ctor;
   _isDefaultDeserializer = isDefaultSerializer(valueDeser);
}

/*
/**********************************************************
/* ContainerDeserializer API
/**********************************************************
 */

@Override
public JavaType getContentType() {
   return _collectionType.getContentType();
}

@SuppressWarnings("unchecked")
@Override
public JsonDeserializer<Object> getContentDeserializer() {
   JsonDeserializer<?> deser = _valueDeserializer;
   return (JsonDeserializer<Object>) deser;
}

/*
/**********************************************************
/* JsonDeserializer API
/**********************************************************
 */

@Override
public Collection<String> deserialize(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   // Ok: must point to START_ARRAY
```

```
    if (!jp.isExpectedStartArrayToken()) {
        throw ctxt.mappingException(_collectionType.getRawClass());
    }

    Collection<String> result;
    try {
        result = (Collection<String>) _defaultCtor.newInstance();
    } catch (Exception e) {
        throw ctxt.instantiationException(_collectionType.getRawClass(), e);
    }
    return deserialize(jp, ctxt, result);
}

@Override
public Collection<String> deserialize(JsonParser jp, DeserializationContext ctxt,
                                      Collection<String> result)
    throws IOException, JsonProcessingException
{
    if (!_isDefaultDeserializer) {
        return deserializeUsingCustom(jp, ctxt, result);
    }
    JsonToken t;

    while ((t = jp.nextToken()) != JsonToken.END_ARRAY) {
        result.add((t == JsonToken.VALUE_NULL) ? null : jp.getText());
    }
    return result;
}

private Collection<String> deserializeUsingCustom(JsonParser jp, DeserializationContext ctxt,
        Collection<String> result)
    throws IOException, JsonProcessingException
{
    JsonToken t;
    final JsonDeserializer<String> deser = _valueDeserializer;

    while ((t = jp.nextToken()) != JsonToken.END_ARRAY) {
        String value;

        if (t == JsonToken.VALUE_NULL) {
            value = null;
        } else {
            value = deser.deserialize(jp, ctxt);
        }
        result.add(value);
    }
    return result;
}
```

```
  @Override
  public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
        TypeDeserializer typeDeserializer)
    throws IOException, JsonProcessingException
  {
    // In future could check current token... for now this should be enough:
    return typeDeserializer.deserializeTypedFromArray(jp, ctxt);
  }

}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.util.*;

import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.TypeDeserializer;

/**
*
* <p>
* Note: casting within this class is all messed up -- just could not figure out a way
* to properly deal with recursive definition of "EnumSet<K extends Enum<K>, V>
*
* @author tsaloranta
*/
public final class EnumSetDeserializer
  extends StdDeserializer<EnumSet<?>>
{
  @SuppressWarnings("unchecked")
 final Class<Enum> _enumClass;

  final EnumDeserializer _enumDeserializer;

  @SuppressWarnings("unchecked")
 public EnumSetDeserializer(EnumResolver enumRes)
  {
    super(EnumSet.class);
    _enumDeserializer = new EnumDeserializer(enumRes);
    // this is fugly, but not sure of a better way...
    _enumClass = (Class<Enum>) ((Class<?>) enumRes.getEnumClass());
  }

  @SuppressWarnings("unchecked")
```

```
  @Override
  public EnumSet<?> deserialize(JsonParser jp, DeserializationContext ctxt)
     throws IOException, JsonProcessingException
  {
     // Ok: must point to START_ARRAY (or equivalent)
     if (!jp.isExpectedStartArrayToken()) {
        throw ctxt.mappingException(EnumSet.class);
     }
     EnumSet result = constructSet();
     JsonToken t;

     while ((t = jp.nextToken()) != JsonToken.END_ARRAY) {
        /* What to do with nulls? Fail or ignore? Fail, for now
         * (note: would fail if we passed it to EnumDeserializer, too,
         * but in general nulls should never be passed to non-container
         * deserializers)
         */
        if (t == JsonToken.VALUE_NULL) {
           throw ctxt.mappingException(_enumClass);
        }
        Enum<?> value = _enumDeserializer.deserialize(jp, ctxt);
        result.add(value);
     }
     return result;
  }

  @Override
  public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
         TypeDeserializer typeDeserializer)
     throws IOException, JsonProcessingException
  {
     return typeDeserializer.deserializeTypedFromArray(jp, ctxt);
  }

  @SuppressWarnings("unchecked")
  private EnumSet constructSet()
  {
   // superbly ugly... but apparently necessary
   return EnumSet.noneOf(_enumClass);
  }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.lang.reflect.*;
import java.util.*;

import org.codehaus.jackson.*;
```

```
import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.introspect.AnnotatedConstructor;
import org.codehaus.jackson.map.introspect.AnnotatedMember;
import org.codehaus.jackson.map.introspect.AnnotatedMethod;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.map.util.ClassUtil;

/**
* Container for different kinds of Creators; objects capable of instantiating
* (and possibly partially or completely initializing) POJOs.
*/
abstract class Creator
{
  // Class only used for namespacing, not as base class
  private Creator() { }

  /**
   * Creator implementation that can handle simple deserialization from
   * Json String values.
   */
  final static class StringBased
  {
     protected final Class<?> _valueClass;
     protected final Method _factoryMethod;
     protected final Constructor<?> _ctor;

     public StringBased(Class<?> valueClass, AnnotatedConstructor ctor,
                        AnnotatedMethod factoryMethod)
     {
        _valueClass = valueClass;
        _ctor = (ctor == null) ? null : ctor.getAnnotated();
        _factoryMethod = (factoryMethod == null) ? null : factoryMethod.getAnnotated();
     }

     public Object construct(String value)
     {
        try {
           if (_ctor != null) {
              return _ctor.newInstance(value);
           }
           if (_factoryMethod != null) {
              return _factoryMethod.invoke(_valueClass, value);
           }
        } catch (Exception e) {
           ClassUtil.unwrapAndThrowAsIAE(e);
        }
        return null;
```

```
    }
}

/**
 * Creator implementation that can handle simple deserialization from
 * Json Number values.
 */
final static class NumberBased
{
    protected final Class<?> _valueClass;

    protected final Constructor<?> _intCtor;
    protected final Constructor<?> _longCtor;

    protected final Method _intFactoryMethod;
    protected final Method _longFactoryMethod;

    public NumberBased(Class<?> valueClass,
                AnnotatedConstructor intCtor, AnnotatedMethod ifm,
                AnnotatedConstructor longCtor, AnnotatedMethod lfm)
    {
        _valueClass = valueClass;
        _intCtor = (intCtor == null) ? null : intCtor.getAnnotated();
        _longCtor = (longCtor == null) ? null : longCtor.getAnnotated();
        _intFactoryMethod = (ifm == null) ? null : ifm.getAnnotated();
        _longFactoryMethod = (lfm == null) ? null : lfm.getAnnotated();
    }

    public Object construct(int value)
    {
        // First: "native" int methods work best:
        try {
            if (_intCtor != null) {
                return _intCtor.newInstance(value);
            }
            if (_intFactoryMethod != null) {
                return _intFactoryMethod.invoke(_valueClass, Integer.valueOf(value));
            }
        } catch (Exception e) {
            ClassUtil.unwrapAndThrowAsIAE(e);
        }
        // but if not, can do widening conversion
        return construct((long) value);
    }

    public Object construct(long value)
    {
        /* For longs we don't even try casting down to ints;
```

```
         * theoretically could try if value fits... but let's
         * leave that as a future improvement
         */
        try {
            if (_longCtor != null) {
                return _longCtor.newInstance(value);
            }
            if (_longFactoryMethod != null) {
                return _longFactoryMethod.invoke(_valueClass, Long.valueOf(value));
            }
        } catch (Exception e) {
            ClassUtil.unwrapAndThrowAsIAE(e);
        }
        return null;
    }
}

/**
 * Creator implementation used for cases where parts of deserialization
 * are delegated to another serializer, by first binding to an intermediate
 * object, and then passing that object to the delegate creator (and
 * then deserializer).
 */
final static class Delegating
{
    /**
     * Annotated creator object (single-argument constructor or
     * single-argument static method) that is used for instantation;
     * as well as for providing contextual information.
     */
    protected final AnnotatedMember _creator;

/**
 * Type to deserialize JSON to, as well as the type to pass to
 * creator (constructor, factory method)
 */
protected final JavaType _valueType;

    protected final Constructor<?> _ctor;
    protected final Method _factoryMethod;

/**
 * Delegate deserializer to use for actual deserialization, before
 * instantiating value
 */
protected JsonDeserializer<Object> _deserializer;

    public Delegating(AnnotatedConstructor ctor, AnnotatedMethod factory)
```

```
{
        if (ctor != null) {
            _creator = ctor;
            _ctor = ctor.getAnnotated();
            _factoryMethod = null;
            _valueType = TypeFactory.type(ctor.getParameterType(0));
        } else if (factory != null) {
            _creator = factory;
            _ctor = null;
            _factoryMethod = factory.getAnnotated();
            _valueType = TypeFactory.type(factory.getParameterType(0));
        } else {
            throw new IllegalArgumentException("Internal error: neither delegating constructor nor factory method
passed");
        }
 }

 public JavaType getValueType() { return _valueType; }

 public AnnotatedMember getCreator() { return _creator; }

 public void setDeserializer(JsonDeserializer<Object> deser)
 {
    _deserializer = deser;
 }

 public Object deserialize(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
 {
    Object value = _deserializer.deserialize(jp, ctxt);
        try {
            if (_ctor != null) {
                return _ctor.newInstance(value);
            }
            // static method, 'obj' can be null
 return _factoryMethod.invoke(null, value);
        } catch (Exception e) {
            ClassUtil.unwrapAndThrowAsIAE(e);
 return null;
        }
 }
  }

  /**
   * Creator implementation used to handle details of using a "non-default"
   * creator (constructor or factory that takes one or more arguments
   * that represent logical bean properties)
   */
```

```
  final static class PropertyBased
  {
     protected final Constructor<?> _ctor;
     protected final Method _factoryMethod;

     /**
      * Map that contains property objects for either constructor or factory
      * method (whichever one is null: one property for each
      * parameter for that one), keyed by logical property name
      */
     protected final HashMap<String, SettableBeanProperty> _properties;

     /**
      * If some property values must always have a non-null value (like
      * primitive types do), this array contains such default values.
      */
     protected final Object[]  _defaultValues;

     public PropertyBased(AnnotatedConstructor ctor, SettableBeanProperty[] ctorProps,
           AnnotatedMethod factory, SettableBeanProperty[] factoryProps)
     {
        // We will only use one: and constructor has precedence over factory
        SettableBeanProperty[] props;
        if (ctor != null) {
           _ctor = ctor.getAnnotated();
           _factoryMethod = null;
           props = ctorProps;
        } else if (factory != null) {
           _ctor = null;
           _factoryMethod = factory.getAnnotated();
           props = factoryProps;
        } else {
           throw new IllegalArgumentException("Internal error: neither delegating constructor nor factory method
passed");
        }
        _properties = new HashMap<String, SettableBeanProperty>();
        // [JACKSON-372]: primitive types need extra care
        Object[] defValues = null;
        for (int i = 0, len = props.length; i < len; ++i) {
           SettableBeanProperty prop = props[i];
           _properties.put(prop.getName(), prop);
           if (prop.getType().isPrimitive()) {
              if (defValues == null) {
                 defValues = new Object[len];
              }
              defValues[i] = ClassUtil.defaultValue(prop.getType().getRawClass());
           }
        }
```

```
        _defaultValues = defValues;
    }

    public Collection<SettableBeanProperty> properties() {
        return _properties.values();
    }

    public SettableBeanProperty findCreatorProperty(String name) {
        return _properties.get(name);
    }

    /**
     * Method called when starting to build a bean instance.
     */
    public PropertyValueBuffer startBuilding(JsonParser jp, DeserializationContext ctxt)
    {
        return new PropertyValueBuffer(jp, ctxt, _properties.size());
    }

    public Object build(PropertyValueBuffer buffer)
        throws Exception
    {
        Object bean;
        try {
            if (_ctor != null) {
                bean = _ctor.newInstance(buffer.getParameters(_defaultValues));
            } else {
                bean =  _factoryMethod.invoke(null, buffer.getParameters(_defaultValues));
            }
        } catch (Exception e) {
            ClassUtil.throwRootCause(e);
            return null; // never gets here
        }
        // Anything buffered?
        for (PropertyValue pv = buffer.buffered(); pv != null; pv = pv.next) {
            pv.assign(bean);
        }
        return bean;
    }
  }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
```

```
/**
* Deserializer that builds on basic {@link BeanDeserializer} but
* override some aspects like instance construction.
*<p>
* Note that this deserializer was significantly changed in Jackson 1.7
* (due to massive changes in {@link BeanDeserializer}).
*/
public class ThrowableDeserializer
  extends BeanDeserializer
{
  protected final static String PROP_NAME_MESSAGE = "message";

  /*
  /**********************************************************
  /* Construction
  /**********************************************************
   */

  public ThrowableDeserializer(BeanDeserializer baseDeserializer)
  {
     super(baseDeserializer);
  }

  /*
  /**********************************************************
  /* Overridden methods
  /**********************************************************
   */

  @Override
  public Object deserializeFromObject(JsonParser jp, DeserializationContext ctxt)
     throws IOException, JsonProcessingException
  {
     // 30-Sep-2010, tatu: Need to allow use of @JsonCreator, so:
     if (_propertyBasedCreator != null) { // proper @JsonCreator
        return _deserializeUsingPropertyBased(jp, ctxt);
     }
     if (_delegatingCreator != null) { // delegate based one (single-arg, no property name)
        return _delegatingCreator.deserialize(jp, ctxt);
     }
     if (_beanType.isAbstract()) { // for good measure, check this too
        throw JsonMappingException.from(jp, "Can not instantiate abstract type "+_beanType
              +" (need to add/enable type information?)");
     }
     // and finally, verify we do have single-String arg constructor (if no @JsonCreator)
     if (_stringCreator == null) {
        throw new JsonMappingException("Can not deserialize Throwable of type "+_beanType
              +" without having either single-String-arg constructor; or explicit @JsonCreator");
```

```
}

Object throwable = null;
Object[] pending = null;
int pendingIx = 0;

for (; jp.getCurrentToken() != JsonToken.END_OBJECT; jp.nextToken()) {
   String propName = jp.getCurrentName();
   SettableBeanProperty prop = _beanProperties.find(propName);
   jp.nextToken(); // to point to field value

   if (prop != null) { // normal case
      if (throwable != null) {
         prop.deserializeAndSet(jp, ctxt, throwable);
         continue;
      }
      // nope; need to defer
      if (pending == null) {
         int len = _beanProperties.size();
         pending = new Object[len + len];
      }
      pending[pendingIx++] = prop;
      pending[pendingIx++] = prop.deserialize(jp, ctxt);
      continue;
   }

   // Maybe it's "message"?
   if (PROP_NAME_MESSAGE.equals(propName)) {
      throwable = _stringCreator.construct(jp.getText());
      // any pending values?
      if (pending != null) {
         for (int i = 0, len = pendingIx; i < len; i += 2) {
            prop = (SettableBeanProperty)pending[i];
            prop.set(throwable, pending[i+1]);
         }
         pending = null;
      }
      continue;
   }
   /* As per [JACKSON-313], things marked as ignorable should not be
    * passed to any setter
    */
   if (_ignorableProps != null && _ignorableProps.contains(propName)) {
      jp.skipChildren();
      continue;
   }
   if (_anySetter != null) {
      _anySetter.deserializeAndSet(jp, ctxt, throwable, propName);
```

```
            continue;
        }
        // Unknown: let's call handler method
        handleUnknownProperty(jp, ctxt, throwable, propName);
      }
      // Sanity check: did we find "message"?
      if (throwable == null) {
        /* 15-Oct-2010, tatu: Can't assume missing message is an error, since it may be
         *   suppressed during serialization, as per [JACKSON-388].
         *
         *   Should probably allow use of default constructor, too...
         */
        //throw new JsonMappingException("No 'message' property found: could not deserialize "+_beanType);
        throwable = _stringCreator.construct(null);
        // any pending values?
        if (pending != null) {
          for (int i = 0, len = pendingIx; i < len; i += 2) {
            SettableBeanProperty prop = (SettableBeanProperty)pending[i];
            prop.set(throwable, pending[i+1]);
          }
        }
      }
      return throwable;
    }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.util.*;

import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.util.ArrayBuilders;
import org.codehaus.jackson.type.JavaType;

/**
* Basic serializer that can take Json "Object" structure and
* construct a {@link java.util.Map} instance, with typed contents.
*<p>
* Note: for untyped content (one indicated by passing Object.class
* as the type), {@link UntypedObjectDeserializer} is used instead.
* It can also construct {@link java.util.Map}s, but not with specific
* POJO types, only other containers and primitives/wrappers.
```

```
*/
@JacksonStdImpl
public class MapDeserializer
  extends ContainerDeserializer<Map<Object,Object>>
  implements ResolvableDeserializer
{
  // // Configuration: typing, deserializers

  final protected JavaType _mapType;

  /**
   * Key deserializer used, if not null. If null, String from JSON
   * content is used as is.
   */
  final protected KeyDeserializer _keyDeserializer;

  /**
   * Value deserializer.
   */
  final protected JsonDeserializer<Object> _valueDeserializer;

  /**
   * If value instances have polymorphic type information, this
   * is the type deserializer that can handle it
   */
  final protected TypeDeserializer _valueTypeDeserializer;

  // // Instance construction settings:

  final protected Constructor<Map<Object,Object>> _defaultCtor;

  /**
   * If the Map is to be instantiated using non-default constructor
   * or factory method
   * that takes one or more named properties as argument(s),
   * this creator is used for instantiation.
   */
  protected Creator.PropertyBased _propertyBasedCreator;

  // // Any properties to ignore if seen?

  protected HashSet<String> _ignorableProperties;

  /*
  /**********************************************************
  /* Life-cycle
  /**********************************************************
   */
```

```
public MapDeserializer(JavaType mapType, Constructor<Map<Object,Object>> defCtor,
                KeyDeserializer keyDeser, JsonDeserializer<Object> valueDeser,
                TypeDeserializer valueTypeDeser)
{
   super(Map.class);
   _mapType = mapType;
   _defaultCtor = defCtor;
   _keyDeserializer = keyDeser;
   _valueDeserializer = valueDeser;
   _valueTypeDeserializer = valueTypeDeser;
}

/**
 * Method called to add constructor and/or factory method based
 * creators to be used with Map, instead of default constructor.
 */
public void setCreators(CreatorContainer creators)
{
   _propertyBasedCreator = creators.propertyBasedCreator();
}

public void setIgnorableProperties(String[] ignorable)
{
   _ignorableProperties = (ignorable == null || ignorable.length == 0) ?
      null : ArrayBuilders.arrayToSet(ignorable);
}

/*
/**********************************************************
/* ContainerDeserializer API
/**********************************************************
 */

@Override
public JavaType getContentType() {
   return _mapType.getContentType();
}

@Override
public JsonDeserializer<Object> getContentDeserializer() {
   return _valueDeserializer;
}

/*
/**********************************************************
/* Validation, post-processing
/**********************************************************
```

```
 */

/**
 * Method called to finalize setup of this deserializer,
 * after deserializer itself has been registered. This
 * is needed to handle recursive and transitive dependencies.
 */
public void resolve(DeserializationConfig config, DeserializerProvider provider)
   throws JsonMappingException
{
   // just need to worry about property-based one
   if (_propertyBasedCreator != null) {
      for (SettableBeanProperty prop : _propertyBasedCreator.properties()) {
         prop.setValueDeserializer(findDeserializer(config, provider, prop.getType(), prop));
      }
   }
}

/*
/**********************************************************
/* JsonDeserializer API
/**********************************************************
 */

@Override
public Map<Object,Object> deserialize(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   // Ok: must point to START_OBJECT, FIELD_NAME or END_OBJECT
   JsonToken t = jp.getCurrentToken();
   if (t != JsonToken.START_OBJECT && t != JsonToken.FIELD_NAME && t != JsonToken.END_OBJECT) {
      throw ctxt.mappingException(getMapClass());
   }
   if (_propertyBasedCreator != null) {
      return _deserializeUsingCreator(jp, ctxt);
   }
   Map<Object,Object> result;
   if (_defaultCtor == null) {
      throw ctxt.instantiationException(getMapClass(), "No default constructor found");
   }
   try {
      result = _defaultCtor.newInstance();
   } catch (Exception e) {
      throw ctxt.instantiationException(getMapClass(), e);
   }
   _readAndBind(jp, ctxt, result);
   return result;
}
```

```
@Override
public Map<Object,Object> deserialize(JsonParser jp, DeserializationContext ctxt,
                                      Map<Object,Object> result)
   throws IOException, JsonProcessingException
{
   // Ok: must point to START_OBJECT or FIELD_NAME
   JsonToken t = jp.getCurrentToken();
   if (t != JsonToken.START_OBJECT && t != JsonToken.FIELD_NAME) {
      throw ctxt.mappingException(getMapClass());
   }
   _readAndBind(jp, ctxt, result);
   return result;
}

@Override
public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
      TypeDeserializer typeDeserializer)
   throws IOException, JsonProcessingException
{
   // In future could check current token... for now this should be enough:
   return typeDeserializer.deserializeTypedFromObject(jp, ctxt);
}

/*
/**********************************************************
/* Other public accessors
/**********************************************************
 */

@SuppressWarnings("unchecked")
public final Class<?> getMapClass() { return (Class<Map<Object,Object>>) _mapType.getRawClass(); }

@Override public JavaType getValueType() { return _mapType; }

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

protected final void _readAndBind(JsonParser jp, DeserializationContext ctxt,
                                  Map<Object,Object> result)
   throws IOException, JsonProcessingException
{
   JsonToken t = jp.getCurrentToken();
   if (t == JsonToken.START_OBJECT) {
      t = jp.nextToken();
```

```
    }
    final KeyDeserializer keyDes = _keyDeserializer;
    final JsonDeserializer<Object> valueDes = _valueDeserializer;
    final TypeDeserializer typeDeser = _valueTypeDeserializer;
    for (; t == JsonToken.FIELD_NAME; t = jp.nextToken()) {
        // Must point to field name
        String fieldName = jp.getCurrentName();
        Object key = (keyDes == null) ? fieldName : keyDes.deserializeKey(fieldName, ctxt);
        // And then the value...
        t = jp.nextToken();
        if (_ignorableProperties != null && _ignorableProperties.contains(fieldName)) {
            jp.skipChildren();
            continue;
        }
        // Note: must handle null explicitly here; value deserializers won't
        Object value;
        if (t == JsonToken.VALUE_NULL) {
            value = null;
        } else if (typeDeser == null) {
            value = valueDes.deserialize(jp, ctxt);
        } else {
            value = valueDes.deserializeWithType(jp, ctxt, typeDeser);
        }
        /* !!! 23-Dec-2008, tatu: should there be an option to verify
         *   that there are no duplicate field names? (and/or what
         *   to do, keep-first or keep-last)
         */
        result.put(key, value);
    }
}

@SuppressWarnings("unchecked")
public Map<Object,Object> _deserializeUsingCreator(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    final Creator.PropertyBased creator = _propertyBasedCreator;
    PropertyValueBuffer buffer = creator.startBuilding(jp, ctxt);

    JsonToken t = jp.getCurrentToken();
    if (t == JsonToken.START_OBJECT) {
        t = jp.nextToken();
    }
    final JsonDeserializer<Object> valueDes = _valueDeserializer;
    final TypeDeserializer typeDeser = _valueTypeDeserializer;
    for (; t == JsonToken.FIELD_NAME; t = jp.nextToken()) {
        String propName = jp.getCurrentName();
        t = jp.nextToken(); // to get to value
        if (_ignorableProperties != null && _ignorableProperties.contains(propName)) {
```

```
            jp.skipChildren(); // and skip it (in case of array/object)
            continue;
        }
        // creator property?
        SettableBeanProperty prop = creator.findCreatorProperty(propName);
        if (prop != null) {
            // Last property to set?
            Object value = prop.deserialize(jp, ctxt);
            if (buffer.assignParameter(prop.getCreatorIndex(), value)) {
                jp.nextToken();
                Map<Object,Object> result;
                try {
                    result = (Map<Object,Object>)creator.build(buffer);
                } catch (Exception e) {
                    wrapAndThrow(e, _mapType.getRawClass());
                    return null;
                }
                _readAndBind(jp, ctxt, result);
                return result;
            }
            continue;
        }
        // other property? needs buffering
        String fieldName = jp.getCurrentName();
        Object key = (_keyDeserializer == null) ? fieldName : _keyDeserializer.deserializeKey(fieldName, ctxt);
        Object value;
        if (t == JsonToken.VALUE_NULL) {
            value = null;
        } else if (typeDeser == null) {
            value = valueDes.deserialize(jp, ctxt);
        } else {
            value = valueDes.deserializeWithType(jp, ctxt, typeDeser);
        }
        buffer.bufferMapProperty(key, value);
    }
    // end of JSON object?
    // if so, can just construct and leave...
    try {
        return (Map<Object,Object>)creator.build(buffer);
    } catch (Exception e) {
        wrapAndThrow(e, _mapType.getRawClass());
        return null;
    }
}

// note: copied form BeanDeserializer; should try to share somehow...
protected void wrapAndThrow(Throwable t, Object ref)
    throws IOException
```

```
  {
    // to handle StackOverflow:
    while (t instanceof InvocationTargetException && t.getCause() != null) {
      t = t.getCause();
    }
    // Errors and "plain" IOExceptions to be passed as is
    if (t instanceof Error) {
      throw (Error) t;
    }
    // ... except for mapping exceptions
    if (t instanceof IOException && !(t instanceof JsonMappingException)) {
      throw (IOException) t;
    }
    throw JsonMappingException.wrapWithPath(t, ref, null);
  }

}
package org.codehaus.jackson.map.deser;

import java.io.IOException;

import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.TypeDeserializer;

/**
* Base class for deserializers that handle types that are serialized
* as JSON scalars (non-structured, i.e. non-Object, non-Array, values).
*
* @author tatu
*/
public abstract class StdScalarDeserializer<T> extends StdDeserializer<T>
{
  protected StdScalarDeserializer(Class<?> vc) {
    super(vc);
  }

  @Override
  public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
      TypeDeserializer typeDeserializer)
    throws IOException, JsonProcessingException
  {
    return typeDeserializer.deserializeTypedFromScalar(jp, ctxt);
  }
}
package org.codehaus.jackson.map.deser;
```

```
import org.codehaus.jackson.map.AnnotationIntrospector;

import java.util.*;

/**
* Helper class used to resolve String values (either Json Object field
* names or regular String values) into Java Enum instances.
*/
public final class EnumResolver<T extends Enum<T>>
{
  protected final Class<T> _enumClass;

  protected final T[] _enums;

  protected final HashMap<String, T> _enumsById;

  private EnumResolver(Class<T> enumClass, T[] enums, HashMap<String, T> map)
  {
    _enumClass = enumClass;
    _enums = enums;
    _enumsById = map;
  }

  /**
   * Factory method for constructing resolver that maps from Enum.name() into
   * Enum value
   */
  public static <ET extends Enum<ET>> EnumResolver<ET> constructFor(Class<ET> enumCls,
AnnotationIntrospector ai)
  {
    ET[] enumValues = enumCls.getEnumConstants();
    if (enumValues == null) {
      throw new IllegalArgumentException("No enum constants for class "+enumCls.getName());
    }
    HashMap<String, ET> map = new HashMap<String, ET>();
    for (ET e : enumValues) {
      map.put(ai.findEnumValue(e), e);
    }
    return new EnumResolver<ET>(enumCls, enumValues, map);
  }

  /**
   * Factory method for constructing resolver that maps from Enum.toString() into
   * Enum value
   *
   * @since 1.6
   */
  public static <ET extends Enum<ET>> EnumResolver<ET> constructUsingToString(Class<ET> enumCls)
```

```
{
    ET[] enumValues = enumCls.getEnumConstants();
    HashMap<String, ET> map = new HashMap<String, ET>();
    // from last to first, so that in case of duplicate values, first wins
    for (int i = enumValues.length; --i >= 0; ) {
        ET e = enumValues[i];
        map.put(e.toString(), e);
    }
    return new EnumResolver<ET>(enumCls, enumValues, map);
}

/**
 * This method is needed because of the dynamic nature of constructing Enum
 * resolvers.
 */
@SuppressWarnings("unchecked")
public static EnumResolver<?> constructUnsafe(Class<?> rawEnumCls, AnnotationIntrospector ai)
{
    /* This is oh so wrong... but at least ugliness is mostly hidden in just
     * this one place.
     */
    Class<Enum> enumCls = (Class<Enum>) rawEnumCls;
    return constructFor(enumCls, ai);
}

/**
 * Method that needs to be used instead of {@link #constructUsingToString}
 * if static type of enum is not known.
 *
 * @since 1.6
 */
@SuppressWarnings("unchecked")
public static EnumResolver<?> constructUnsafeUsingToString(Class<?> rawEnumCls)
{
    // oh so wrong... not much that can be done tho
    Class<Enum> enumCls = (Class<Enum>) rawEnumCls;
    return constructUsingToString(enumCls);
}

public T findEnum(String key)
{
    return _enumsById.get(key);
}

public T getEnum(int index)
{
    if (index < 0 || index >= _enums.length) {
        return null;
```

```
        }
        return _enums[index];
    }

    public Class<T> getEnumClass() { return _enumClass; }

    public int lastValidIndex() { return _enums.length-1; }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.util.*;

import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.DeserializationConfig;
import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.TypeDeserializer;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.util.ObjectBuffer;

/**
* This deserializer is only used if it is necessary to bind content of
* unknown type (or without regular structure) into generic Java container
* types; Lists, Maps, wrappers, nulls and so on.
*/
@JacksonStdImpl
public class UntypedObjectDeserializer
    extends StdDeserializer<Object>
{
    public UntypedObjectDeserializer() { super(Object.class); }

    /*
    /**********************************************************
    /* Deserializer API
    /**********************************************************
     */

    @Override
    public Object deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        switch (jp.getCurrentToken()) {
        // first, simple types:
        case VALUE_STRING:
            return jp.getText();
```

```
    case VALUE_NUMBER_INT:
        /* [JACKSON-100]: caller may want to get all integral values
         * returned as BigInteger, for consistency
         */
        if (ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_INTEGER_FOR_INTS)) {
            return jp.getBigIntegerValue(); // should be optimal, whatever it is
        }
        return jp.getNumberValue(); // should be optimal, whatever it is

    case VALUE_NUMBER_FLOAT:
        /* [JACKSON-72]: need to allow overriding the behavior regarding
         *   which type to use
         */
        if (ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_DECIMAL_FOR_FLOATS)) {
            return jp.getDecimalValue();
        }
        return Double.valueOf(jp.getDoubleValue());

    case VALUE_TRUE:
        return Boolean.TRUE;
    case VALUE_FALSE:
        return Boolean.FALSE;
    case VALUE_EMBEDDED_OBJECT:
        return jp.getEmbeddedObject();

    case VALUE_NULL: // should not get this but...
        return null;

        // Then structured types:

    case START_ARRAY:
        return mapArray(jp, ctxt);

    case START_OBJECT:
    case FIELD_NAME:
        return mapObject(jp, ctxt);

        // and finally, invalid types
    case END_ARRAY:
    case END_OBJECT:
        break;
    }

    throw ctxt.mappingException(Object.class);
}

@Override
public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
```

```
        TypeDeserializer typeDeserializer)
    throws IOException, JsonProcessingException
{
    JsonToken t = jp.getCurrentToken();
    switch (t) {
    // First: does it look like we had type id wrapping of some kind?
    case START_ARRAY:
    case START_OBJECT:
    case FIELD_NAME:
        /* Output can be as JSON Object, Array or scalar: no way to know
         * a this point:
         */
        return typeDeserializer.deserializeTypedFromAny(jp, ctxt);

    /* Otherwise we probably got a "native" type (ones that map
     * naturally and thus do not need or use type ids)
     */
    case VALUE_STRING:
        return jp.getText();

    case VALUE_NUMBER_INT:
        // For [JACKSON-100], see above:
        if (ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_INTEGER_FOR_INTS)) {
            return jp.getBigIntegerValue();
        }
        return jp.getIntValue();

    case VALUE_NUMBER_FLOAT:
        // For [JACKSON-72], see above
        if (ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_DECIMAL_FOR_FLOATS)) {
            return jp.getDecimalValue();
        }
        return Double.valueOf(jp.getDoubleValue());

    case VALUE_TRUE:
        return Boolean.TRUE;
    case VALUE_FALSE:
        return Boolean.FALSE;
    case VALUE_EMBEDDED_OBJECT:
        return jp.getEmbeddedObject();

    case VALUE_NULL: // should not get this far really but...
        return null;
    }
    throw ctxt.mappingException(Object.class);
}

/*
```

```
/**********************************************************
/* Internal methods
/**********************************************************
 */

protected List<Object> mapArray(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   // Minor optimization to handle small lists (default size for ArrayList is 10)
   if (jp.nextToken()  == JsonToken.END_ARRAY) {
      return new ArrayList<Object>(4);
   }
   ObjectBuffer buffer = ctxt.leaseObjectBuffer();
   Object[] values = buffer.resetAndStart();
   int ptr = 0;
   int totalSize = 0;
   do {
      Object value = deserialize(jp, ctxt);
      ++totalSize;
      if (ptr >= values.length) {
         values = buffer.appendCompletedChunk(values);
         ptr = 0;
      }
      values[ptr++] = value;
   } while (jp.nextToken() != JsonToken.END_ARRAY);
   // let's create almost full array, with 1/8 slack
   ArrayList<Object> result = new ArrayList<Object>(totalSize + (totalSize >> 3) + 1);
   buffer.completeAndClearBuffer(values, ptr, result);
   return result;
}

protected Map<String,Object> mapObject(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   JsonToken t = jp.getCurrentToken();
   if (t == JsonToken.START_OBJECT) {
      t = jp.nextToken();
   }
   // 1.6: minor optimization; let's handle 1 and 2 entry cases separately
   if (t != JsonToken.FIELD_NAME) { // and empty one too
      // empty map might work; but caller may want to modify... so better just give small modifiable
      return new LinkedHashMap<String,Object>(4);
   }
   String field1 = jp.getText();
   jp.nextToken();
   Object value1 = deserialize(jp, ctxt);
   if (jp.nextToken() != JsonToken.FIELD_NAME) { // single entry; but we want modifiable
      LinkedHashMap<String, Object> result = new LinkedHashMap<String, Object>(4);
```

```
            result.put(field1, value1);
            return result;
        }
        String field2 = jp.getText();
        jp.nextToken();
        Object value2 = deserialize(jp, ctxt);
        if (jp.nextToken() != JsonToken.FIELD_NAME) {
            LinkedHashMap<String, Object> result = new LinkedHashMap<String, Object>(4);
            result.put(field1, value1);
            result.put(field2, value2);
            return result;
        }
        // And then the general case; default map size is 16
        LinkedHashMap<String, Object> result = new LinkedHashMap<String, Object>();
        result.put(field1, value1);
        result.put(field2, value2);
        do {
            String fieldName = jp.getText();
            jp.nextToken();
            result.put(fieldName, deserialize(jp, ctxt));
        } while (jp.nextToken() != JsonToken.END_OBJECT);
        return result;
    }
}
package org.codehaus.jackson.map.deser;

import java.lang.reflect.Constructor;
import java.util.HashMap;

import org.codehaus.jackson.map.introspect.AnnotatedConstructor;
import org.codehaus.jackson.map.introspect.AnnotatedMethod;
import org.codehaus.jackson.map.util.ClassUtil;

/**
* Container for set of Creators (constructors, factory methods)
*/
public class CreatorContainer
{
    /// Type of bean being created
    final Class<?> _beanClass;
    final boolean _canFixAccess;

    protected Constructor<?> _defaultConstructor;

    AnnotatedMethod _strFactory, _intFactory, _longFactory;
    AnnotatedMethod _delegatingFactory;
    AnnotatedMethod _propertyBasedFactory;
    SettableBeanProperty[] _propertyBasedFactoryProperties = null;
```

```
    AnnotatedConstructor _strConstructor, _intConstructor, _longConstructor;
    AnnotatedConstructor _delegatingConstructor;
    AnnotatedConstructor _propertyBasedConstructor;
    SettableBeanProperty[] _propertyBasedConstructorProperties = null;

    public CreatorContainer(Class<?> beanClass, boolean canFixAccess) {
        _canFixAccess = canFixAccess;
        _beanClass = beanClass;
    }

    /*
    /**********************************************************
    /* Setters
    /**********************************************************
     */

    public void setDefaultConstructor(Constructor<?> ctor) {
        _defaultConstructor = ctor;
    }

    public void addStringConstructor(AnnotatedConstructor ctor) {
        _strConstructor = verifyNonDup(ctor, _strConstructor, "String");
    }
    public void addIntConstructor(AnnotatedConstructor ctor) {
        _intConstructor = verifyNonDup(ctor, _intConstructor, "int");
    }
    public void addLongConstructor(AnnotatedConstructor ctor) {
        _longConstructor = verifyNonDup(ctor, _longConstructor, "long");
    }

    public void addDelegatingConstructor(AnnotatedConstructor ctor) {
        _delegatingConstructor = verifyNonDup(ctor, _delegatingConstructor, "long");
    }

    public void addPropertyConstructor(AnnotatedConstructor ctor, SettableBeanProperty[] properties)
    {
        _propertyBasedConstructor = verifyNonDup(ctor, _propertyBasedConstructor, "property-based");
        // [JACKSON-470] Better ensure we have no duplicate names either...
        if (properties.length > 1) {
            HashMap<String,Integer> names = new HashMap<String,Integer>();
            for (int i = 0, len = properties.length; i < len; ++i) {
                String name = properties[i].getName();
                Integer old = names.put(name, Integer.valueOf(i));
                if (old != null) {
                    throw new IllegalArgumentException("Duplicate creator property \""+name+"\" (index "+old+" vs
"+i+")");
                }
```

```
        }
    }
    _propertyBasedConstructorProperties = properties;
}

public void addStringFactory(AnnotatedMethod factory) {
    _strFactory = verifyNonDup(factory, _strFactory, "String");
}
public void addIntFactory(AnnotatedMethod factory) {
    _intFactory = verifyNonDup(factory, _intFactory, "int");
}
public void addLongFactory(AnnotatedMethod factory) {
    _longFactory = verifyNonDup(factory, _longFactory, "long");
}

public void addDelegatingFactory(AnnotatedMethod factory) {
    _delegatingFactory = verifyNonDup(factory, _delegatingFactory, "long");
}

public void addPropertyFactory(AnnotatedMethod factory, SettableBeanProperty[] properties)
{
    _propertyBasedFactory = verifyNonDup(factory, _propertyBasedFactory, "property-based");
    _propertyBasedFactoryProperties = properties;
}

/*
/**********************************************************
/* Accessors
/**********************************************************
 */

public Constructor<?> getDefaultConstructor() { return _defaultConstructor; }

public Creator.StringBased stringCreator()
{
    if (_strConstructor == null &&  _strFactory == null) {
        return null;
    }
    return new Creator.StringBased(_beanClass, _strConstructor, _strFactory);
}

public Creator.NumberBased numberCreator()
{
    if (_intConstructor == null && _intFactory == null
        && _longConstructor == null && _longFactory == null) {
        return null;
    }
    return new Creator.NumberBased(_beanClass, _intConstructor, _intFactory,
```

```
                                _longConstructor, _longFactory);
    }

    public Creator.Delegating delegatingCreator()
    {
        if (_delegatingConstructor == null && _delegatingFactory == null) {
            return null;
        }
        return new Creator.Delegating(_delegatingConstructor, _delegatingFactory);
    }

    public Creator.PropertyBased propertyBasedCreator()
    {
        if (_propertyBasedConstructor == null && _propertyBasedFactory == null) {
            return null;
        }
        return new Creator.PropertyBased(_propertyBasedConstructor, _propertyBasedConstructorProperties,
                                _propertyBasedFactory, _propertyBasedFactoryProperties);
    }

    /*
    /**********************************************************
    /* Helper methods
    /**********************************************************
     */

    protected AnnotatedConstructor verifyNonDup(AnnotatedConstructor newOne, AnnotatedConstructor oldOne,
            String type)
    {
        if (oldOne != null) {
            throw new IllegalArgumentException("Conflicting "+type+" constructors: already had "+oldOne+",
encountered "+newOne);
        }
        if (_canFixAccess) {
            ClassUtil.checkAndFixAccess(newOne.getAnnotated());
        }
        return newOne;
    }

    protected AnnotatedMethod verifyNonDup(AnnotatedMethod newOne, AnnotatedMethod oldOne,
                                String type)
    {
        if (oldOne != null) {
            throw new IllegalArgumentException("Conflicting "+type+" factory methods: already had "+oldOne+",
encountered "+newOne);
        }
        if (_canFixAccess) {
            ClassUtil.checkAndFixAccess(newOne.getAnnotated());
```

```
        }
        return newOne;
    }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.text.DateFormat;
import java.text.ParseException;
import java.util.*;

import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.exc.UnrecognizedPropertyException;
import org.codehaus.jackson.map.util.ArrayBuilders;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.map.util.LinkedNode;
import org.codehaus.jackson.map.util.ObjectBuffer;
import org.codehaus.jackson.type.JavaType;

/**
* Default implementation of {@link DeserializationContext}.
*/
public class StdDeserializationContext
    extends DeserializationContext
{
    /**
     * Let's limit length of error messages, for cases where underlying data
     * may be very large -- no point in spamming logs with megs of meaningless
     * data.
     */
    final static int MAX_ERROR_STR_LEN = 500;

    // // // Configuration

    /**
     * Currently active parser used for deserialization.
     * May be different from the outermost parser
     * when content is buffered.
     */
    protected JsonParser _parser;

    /**
     * @since 1.5
     */
    protected final DeserializerProvider _deserProvider;
```

```
// // // Helper object recycling

protected ArrayBuilders _arrayBuilders;

protected ObjectBuffer _objectBuffer;

protected DateFormat _dateFormat;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public StdDeserializationContext(DeserializationConfig config, JsonParser jp,
       DeserializerProvider prov)
{
 super(config);
    _parser = jp;
    _deserProvider = prov;
}

/*
/**********************************************************
/* Public API, accessors
/**********************************************************
 */

@Override
public DeserializerProvider getDeserializerProvider() {
   return _deserProvider;
}

/**
 * Method for accessing the currently active parser.
 * May be different from the outermost parser
 * when content is buffered.
 *<p>
 * Use of this method is discouraged: if code has direct access
 * to the active parser, that should be used instead.
 */
@Override
public JsonParser getParser() { return _parser; }

/*
/**********************************************************
/* Public API, helper object recycling
```

```
/**********************************************************
 */

@Override
public final ObjectBuffer leaseObjectBuffer()
{
   ObjectBuffer buf = _objectBuffer;
   if (buf == null) {
      buf = new ObjectBuffer();
   } else {
      _objectBuffer = null;
   }
   return buf;
}

@Override
public final void returnObjectBuffer(ObjectBuffer buf)
{
   /* Already have a reusable buffer? Let's retain bigger one
    * (or if equal, favor newer one, shorter life-cycle)
    */
   if (_objectBuffer == null
      || buf.initialCapacity() >= _objectBuffer.initialCapacity()) {
      _objectBuffer = buf;
   }
}

@Override
public final ArrayBuilders getArrayBuilders()
{
   if (_arrayBuilders == null) {
      _arrayBuilders = new ArrayBuilders();
   }
   return _arrayBuilders;
}

/*
/**********************************************************
/* Parsing methods that may use reusable/recyclable objects
/**********************************************************
 */

@Override
public Date parseDate(String dateStr)
   throws IllegalArgumentException
{
   try {
      return getDateFormat().parse(dateStr);
```

```
    } catch (ParseException pex) {
        throw new IllegalArgumentException(pex.getMessage());
    }
  }

  @Override
  public Calendar constructCalendar(Date d)
  {
    /* 08-Jan-2008, tatu: not optimal, but should work for the
     *   most part; let's revise as needed.
     */
    Calendar c = Calendar.getInstance();
    c.setTime(d);
    return c;
  }
  /*
  /**********************************************************
  /* Public API, problem handling, reporting
  /**********************************************************
   */

  /**
   * Method deserializers can call to inform configured {@link DeserializationProblemHandler}s
   * of an unrecognized property.
   *
   * @since 1.5
   */
  @Override
  public boolean handleUnknownProperty(JsonParser jp, JsonDeserializer<?> deser, Object instanceOrClass, String
propName)
    throws IOException, JsonProcessingException
  {
    LinkedNode<DeserializationProblemHandler> h = _config.getProblemHandlers();
    if (h != null) {
      /* 04-Jan-2009, tatu: Ugh. Need to mess with currently active parser
       *   since parser is not explicitly passed to handler... that was a mistake
       */
      JsonParser oldParser = _parser;
      _parser = jp;
      try {
        while (h != null) {
          // Can bail out if it's handled
          if (h.value().handleUnknownProperty(this, deser, instanceOrClass, propName)) {
            return true;
          }
          h = h.next();
        }
      } finally {
```

```
                _parser = oldParser;
            }
        }
        return false;
    }

    @Override
    public JsonMappingException mappingException(Class<?> targetClass)
    {
        String clsName = _calcName(targetClass);
        return JsonMappingException.from(_parser, "Can not deserialize instance of "+clsName+" out of
"+_parser.getCurrentToken()+" token");
    }

    @Override
    public JsonMappingException instantiationException(Class<?> instClass, Exception e)
    {
        return JsonMappingException.from(_parser, "Can not construct instance of "+instClass.getName()+", problem:
"+e.getMessage());
    }

    @Override
    public JsonMappingException instantiationException(Class<?> instClass, String msg)
    {
        return JsonMappingException.from(_parser, "Can not construct instance of "+instClass.getName()+", problem:
"+msg);
    }

    /**
     * Method that will construct an exception suitable for throwing when
     * some String values are acceptable, but the one encountered is not
     */
    @Override
   public JsonMappingException weirdStringException(Class<?> instClass, String msg)
    {
        return JsonMappingException.from(_parser, "Can not construct instance of "+instClass.getName()+" from
String value '"+_valueDesc()+"': "+msg);
    }

    @Override
    public JsonMappingException weirdNumberException(Class<?> instClass, String msg)
    {
        return JsonMappingException.from(_parser, "Can not construct instance of "+instClass.getName()+" from
number value ("+_valueDesc()+"): "+msg);
    }

    @Override
    public JsonMappingException weirdKeyException(Class<?> keyClass, String keyValue, String msg)
```

```
    {
        return JsonMappingException.from(_parser, "Can not construct Map key of type "+keyClass.getName()+" from
String \""+_desc(keyValue)+"\": "+msg);
    }

    @Override
    public JsonMappingException wrongTokenException(JsonParser jp, JsonToken expToken, String msg)
    {
        return JsonMappingException.from(jp, "Unexpected token ("+jp.getCurrentToken()+"), expected
"+expToken+": "+msg);
    }

    @Override
    public JsonMappingException unknownFieldException(Object instanceOrClass, String fieldName)
    {
        return UnrecognizedPropertyException.from(_parser, instanceOrClass, fieldName);
    }

    @Override
    public JsonMappingException unknownTypeException(JavaType type, String id)
    {
        return JsonMappingException.from(_parser, "Could not resolve type id '"+id+"' into a subtype of "+type);
    }

    /*
    /**********************************************************
    /* Overridable internal methods
    /**********************************************************
     */

    protected DateFormat getDateFormat()
    {
        if (_dateFormat == null) {
            // must create a clone since Formats are not thread-safe:
            _dateFormat = (DateFormat)_config.getDateFormat().clone();
        }
        return _dateFormat;
    }

    protected String determineClassName(Object instance)
    {
        return ClassUtil.getClassDescription(instance);
    }

    /*
    /**********************************************************
    /* Other internal methods
    /**********************************************************
```

```
     */

    protected String _calcName(Class<?> cls)
    {
        if (cls.isArray()) {
            return _calcName(cls.getComponentType())+"[]";
        }
        return cls.getName();
    }

    protected String _valueDesc()
    {
        try {
            return _desc(_parser.getText());
        } catch (Exception e) {
            return "[N/A]";
        }
    }
    protected String _desc(String desc)
    {
        // !!! should we quote it? (in case there are control chars, linefeeds)
        if (desc.length() > MAX_ERROR_STR_LEN) {
            desc = desc.substring(0, MAX_ERROR_STR_LEN) + "]...[" + desc.substring(desc.length() -
MAX_ERROR_STR_LEN);
        }
        return desc;
    }
}
package org.codehaus.jackson.map.deser;

import java.io.*;
import java.net.InetAddress;
import java.net.URI;
import java.net.URL;
import java.util.*;
import java.util.regex.Pattern;

import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.DeserializationContext;

/**
* Base class for simple deserializer which only accept JSON String
* values as the source.
*/
public abstract class FromStringDeserializer<T>
    extends StdScalarDeserializer<T>
```

```
{
  protected FromStringDeserializer(Class<?> vc) {
    super(vc);
  }

  public static Iterable<FromStringDeserializer<?>>all()
  {
    ArrayList<FromStringDeserializer<?>> all = new ArrayList<FromStringDeserializer<?>>();

    all.add(new UUIDDeserializer());
    all.add(new URLDeserializer());
    all.add(new URIDeserializer());
    all.add(new CurrencyDeserializer());
    all.add(new PatternDeserializer());
    // since 1.7:
    all.add(new LocaleDeserializer());
    // 1.8:
    all.add(new InetAddressDeserializer());
    all.add(new TimeZoneDeserializer());

    return all;
  }

  @SuppressWarnings("unchecked")
  @Override
  public final T deserialize(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
    if (jp.getCurrentToken() == JsonToken.VALUE_STRING) {
      String text = jp.getText().trim();
      // 15-Oct-2010, tatu: Empty String usually means null, so
      if (text.length() == 0) {
        return null;
      }
      try {
        T result = _deserialize(text, ctxt);
        if (result != null) {
          return result;
        }
      } catch (IllegalArgumentException iae) {
        // nothing to do here, yet? We'll fail anyway
      }
      throw ctxt.weirdStringException(_valueClass, "not a valid textual representation");
    }
    if (jp.getCurrentToken() == JsonToken.VALUE_EMBEDDED_OBJECT) {
      // Trivial cases; null to null, instance of type itself returned as is
      Object ob = jp.getEmbeddedObject();
      if (ob == null) {
```

```
            return null;
        }
        if (_valueClass.isAssignableFrom(ob.getClass())) {
            return (T) ob;
        }
        return _deserializeEmbedded(ob, ctxt);
    }
    throw ctxt.mappingException(_valueClass);
}

protected abstract T _deserialize(String value, DeserializationContext ctxt)
    throws IOException, JsonProcessingException;

protected T _deserializeEmbedded(Object ob, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    // default impl: error out
    throw ctxt.mappingException("Don't know how to convert embedded Object of type "
            +ob.getClass().getName()+" into "+_valueClass.getName());
}

/*
/**********************************************************
/* Then concrete implementations
/**********************************************************
 */

public static class UUIDDeserializer
    extends FromStringDeserializer<UUID>
{
    public UUIDDeserializer() { super(UUID.class); }

    @Override
    protected UUID _deserialize(String value, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        return UUID.fromString(value);
    }

    @Override
    protected UUID _deserializeEmbedded(Object ob, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        if (ob instanceof byte[]) {
            byte[] bytes = (byte[]) ob;
            if (bytes.length != 16) {
                ctxt.mappingException("Can only construct UUIDs from 16 byte arrays; got "+bytes.length+" bytes");
            }
```

```
            // clumsy, but should work for now...
            DataInputStream in = new DataInputStream(new ByteArrayInputStream(bytes));
            long l1 = in.readLong();
            long l2 = in.readLong();
            return new UUID(l1, l2);
        }
        super._deserializeEmbedded(ob, ctxt);
        return null; // never gets here
    }
}

public static class URLDeserializer
    extends FromStringDeserializer<URL>
{
    public URLDeserializer() { super(URL.class); }

    @Override
    protected URL _deserialize(String value, DeserializationContext ctxt)
        throws IOException
    {
        return new URL(value);
    }
}

public static class URIDeserializer
    extends FromStringDeserializer<URI>
{
    public URIDeserializer() { super(URI.class); }

    @Override
    protected URI _deserialize(String value, DeserializationContext ctxt)
        throws IllegalArgumentException
    {
        return URI.create(value);
    }
}

public static class CurrencyDeserializer
    extends FromStringDeserializer<Currency>
{
    public CurrencyDeserializer() { super(Currency.class); }

    @Override
    protected Currency _deserialize(String value, DeserializationContext ctxt)
        throws IllegalArgumentException
    {
        // will throw IAE if unknown:
        return Currency.getInstance(value);
```

```
    }
}

public static class PatternDeserializer
    extends FromStringDeserializer<Pattern>
{
    public PatternDeserializer() { super(Pattern.class); }

    @Override
    protected Pattern _deserialize(String value, DeserializationContext ctxt)
        throws IllegalArgumentException
    {
        // will throw IAE (or its subclass) if malformed
        return Pattern.compile(value);
    }
}

/**
 * Kept protected as it's not meant to be extensible at this point
 *
 * @since 1.7
 */
protected static class LocaleDeserializer
    extends FromStringDeserializer<Locale>
{
    public LocaleDeserializer() { super(Locale.class); }

    @Override
    protected Locale _deserialize(String value, DeserializationContext ctxt)
        throws IOException
    {
        int ix = value.indexOf('_');
        if (ix < 0) { // single argument
            return new Locale(value);
        }
        String first = value.substring(0, ix);
        value = value.substring(ix+1);
        ix = value.indexOf('_');
        if (ix < 0) { // two pieces
            return new Locale(first, value);
        }
        String second = value.substring(0, ix);
        return new Locale(first, second, value.substring(ix+1));
    }
}

/**
 * As per [JACKSON-484], also need special handling for InetAddress...
```

```
     *
     * @since 1.7.4
     */
    protected static class InetAddressDeserializer
        extends FromStringDeserializer<InetAddress>
    {
        public InetAddressDeserializer() { super(InetAddress.class); }

        @Override
        protected InetAddress _deserialize(String value, DeserializationContext ctxt)
            throws IOException
        {
            return InetAddress.getByName(value);
        }
    }

    /**
     * As per [JACKSON-522], also need special handling for InetAddress...
     *
     * @since 1.7.4
     */
    protected static class TimeZoneDeserializer
        extends FromStringDeserializer<TimeZone>
    {
        public TimeZoneDeserializer() { super(TimeZone.class); }

        @Override
        protected TimeZone _deserialize(String value, DeserializationContext ctxt)
            throws IOException
        {
            return TimeZone.getTimeZone(value);
        }
    }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.lang.annotation.Annotation;
import java.lang.reflect.*;
import java.util.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.introspect.AnnotatedField;
import org.codehaus.jackson.map.introspect.AnnotatedMember;
import org.codehaus.jackson.map.introspect.AnnotatedMethod;
import org.codehaus.jackson.map.introspect.AnnotatedParameter;
import org.codehaus.jackson.map.util.Annotations;
```

```
import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.util.InternCache;

/**
* Base class for settable properties of a bean: contains
* both type and name definitions, and reflection-based set functionality.
* Concrete sub-classes implement details, so that both field- and
* setter-backed properties can be handled
*/
public abstract class SettableBeanProperty
  implements BeanProperty // since 1.7
{
  /**
   * Logical name of the property (often but not always derived
   * from the setter method name)
   */
  protected final String _propName;

  /**
   * Base type for property; may be a supertype of actual value.
   */
  protected final JavaType _type;

  /**
   * Class that contains this property (either class that declares
   * the property or one of its subclasses), class that is
   * deserialized using deserializer that contains this property.
   */
  protected final Annotations _contextAnnotations;

  /**
   * Deserializer used for handling property value.
   */
  protected JsonDeserializer<Object> _valueDeserializer;

  /**
   * If value will contain type information (to support
   * polymorphic handling), this is the type deserializer
   * used to handle type resolution.
   */
  protected TypeDeserializer _valueTypeDeserializer;

  /**
   * Object used to figure out value to be used when 'null' literal is encountered in JSON.
   * For most types simply Java null, but for primitive types must
   * be a non-null value (like Integer.valueOf(0) for int).
   *
   * @since 1.7
```

```
     */
    protected NullProvider _nullProvider;

    /**
     * If property represents a managed (forward) reference
     * (see [JACKSON-235]), we will need name of reference for
     * later linking.
     */
    protected String _managedReferenceName;

    /**
     * Index of property (within all property of a bean); assigned
     * when all properties have been collected. Order of entries
     * is arbitrary, but once indexes are assigned they are not
     * changed.
     *
     * @since 1.7
     */
    protected int _propertyIndex = -1;

    /*
    /**********************************************************
    /* Life-cycle (construct & configure)
    /**********************************************************
     */

    protected SettableBeanProperty(String propName, JavaType type, TypeDeserializer typeDeser,
            Annotations contextAnnotations)
    {
        /* 09-Jan-2009, tatu: Intern()ing makes sense since Jackson parsed
         *   field names are (usually) interned too, hence lookups will be faster.
         */
        // 23-Oct-2009, tatu: should this be disabled wrt [JACKSON-180]?
        if (propName == null || propName.length() == 0) {
            _propName = "";
        } else {
            _propName = InternCache.instance.intern(propName);
        }
        _type = type;
        _contextAnnotations = contextAnnotations;
        _valueTypeDeserializer = typeDeser;
    }

    public void setValueDeserializer(JsonDeserializer<Object> deser)
    {
        if (_valueDeserializer != null) { // sanity check
            throw new IllegalStateException("Already had assigned deserializer for property "'+getName()+"' (class
    "+getDeclaringClass().getName()+")");
```

```
        }
        _valueDeserializer = deser;
        Object nvl = _valueDeserializer.getNullValue();
        _nullProvider = (nvl == null) ? null : new NullProvider(_type, nvl);
    }

    public void setManagedReferenceName(String n) {
        _managedReferenceName = n;
    }

    /**
     * Method used to assign index for property.
     *
     * @since 1.7
     */
    public void assignIndex(int index) {
        if (_propertyIndex != -1) {
            throw new IllegalStateException("Property '"+getName()+"' already had index ("+_propertyIndex+"), trying
to assign "+index);
        }
        _propertyIndex = index;
    }

    /*
    /**********************************************************
    /* BeanProperty impl
    /**********************************************************
     */

    public final String getName() { return _propName; }

    public JavaType getType() { return _type; }

    public abstract <A extends Annotation> A getAnnotation(Class<A> acls);

    public abstract AnnotatedMember getMember();

    public <A extends Annotation> A getContextAnnotation(Class<A> acls)
    {
        return _contextAnnotations.get(acls);
    }

    /*
    /**********************************************************
    /* Accessors
    /**********************************************************
     */
```

```
protected final Class<?> getDeclaringClass() {
   return getMember().getDeclaringClass();
}

/**
 * @deprecated Since 1.7, use {@link #getName} instead.
 */
@Deprecated
public String getPropertyName() { return _propName; }

public String getManagedReferenceName() { return _managedReferenceName; }

public boolean hasValueDeserializer() { return (_valueDeserializer != null); }

/**
 * Method to use for accessing index of the property (related to
 * other properties in the same context); currently only applicable
 * to "Creator properties".
 *<p>
 * Base implementation returns -1 to indicate that no index exists
 * for the property.
 */
public int getCreatorIndex() { return -1; }

/**
 * Method for accessing unique index of this property; indexes are
 * assigned once all properties of a {@link BeanDeserializer} have
 * been collected.
 *
 * @return Index of this property
 *
 * @since 1.7
 */
public int getProperytIndex() { return _propertyIndex; }

/*
/**********************************************************
/* Public API
/**********************************************************
 */

/**
 * Method called to deserialize appropriate value, given parser (and
 * context), and set it using appropriate mechanism.
 * Pre-condition is that passed parser must point to the first token
 * that should be consumed to produce the value (the only value for
 * scalars, multiple for Objects and Arrays).
 */
```

```
public abstract void deserializeAndSet(JsonParser jp, DeserializationContext ctxt,
                                       Object instance)
   throws IOException, JsonProcessingException;

public abstract void set(Object instance, Object value)
   throws IOException;

/**
 * This method is needed by some specialized bean deserializers,
 * and also called by some {@link #deserializeAndSet} implementations.
 *<p>
 * Pre-condition is that passed parser must point to the first token
 * that should be consumed to produce the value (the only value for
 * scalars, multiple for Objects and Arrays).
 */
public final Object deserialize(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   JsonToken t = jp.getCurrentToken();
   if (t == JsonToken.VALUE_NULL) {
      return (_nullProvider == null) ? null : _nullProvider.nullValue(ctxt);
   }
   if (_valueTypeDeserializer != null) {
      return _valueDeserializer.deserializeWithType(jp, ctxt, _valueTypeDeserializer);
   }
   return _valueDeserializer.deserialize(jp, ctxt);
}

/*
/**********************************************************
/* Helper methods
/**********************************************************
 */

/**
 * Method that takes in exception of any type, and casts or wraps it
 * to an IOException or its subclass.
 */
protected void _throwAsIOE(Exception e, Object value)
   throws IOException
{
   if (e instanceof IllegalArgumentException) {
      String actType = (value == null) ? "[NULL]" : value.getClass().getName();
      StringBuilder msg = new StringBuilder("Problem deserializing property '").append(getPropertyName());
      msg.append("' (expected type: ").append(getType());
      msg.append("; actual type: ").append(actType).append(")");
      String origMsg = e.getMessage();
      if (origMsg != null) {
```

```
            msg.append(", problem: ").append(origMsg);
        } else {
            msg.append(" (no error message provided)");
        }
        throw new JsonMappingException(msg.toString(), null, e);
    }
    _throwAsIOE(e);
}

protected IOException _throwAsIOE(Exception e)
    throws IOException
{
    if (e instanceof IOException) {
        throw (IOException) e;
    }
    if (e instanceof RuntimeException) {
        throw (RuntimeException) e;
    }
    // let's wrap the innermost problem
    Throwable th = e;
    while (th.getCause() != null) {
        th = th.getCause();
    }
    throw new JsonMappingException(th.getMessage(), null, th);
}

@Override public String toString() { return "[property '"+getName()+"']"; }

/*
/**********************************************************
/* Impl classes
/**********************************************************
 */

/**
 * This concrete sub-class implements property that is set
 * using regular "setter" method.
 */
public final static class MethodProperty
    extends SettableBeanProperty
{
    protected final AnnotatedMethod _annotated;

    /**
     * Setter method for modifying property value; used for
     * "regular" method-accessible properties.
     */
    protected final Method _setter;
```

```
    public MethodProperty(String name, JavaType type, TypeDeserializer typeDeser,
            Annotations contextAnnotations, AnnotatedMethod method)
    {
        super(name, type, typeDeser, contextAnnotations);
        _annotated = method;
        _setter = method.getAnnotated();
    }

    /*
    /**********************************************************
    /* BeanProperty impl
    /**********************************************************
     */

    @Override
    public <A extends Annotation> A getAnnotation(Class<A> acls) {
        return _annotated.getAnnotation(acls);
    }

    @Override public AnnotatedMember getMember() {  return _annotated; }

    /*
    /**********************************************************
    /* Overridden methods
    /**********************************************************
     */

    @Override
    public void deserializeAndSet(JsonParser jp, DeserializationContext ctxt,
            Object instance)
        throws IOException, JsonProcessingException
    {
        set(instance, deserialize(jp, ctxt));
    }

    @Override
    public final void set(Object instance, Object value)
        throws IOException
    {
        try {
            _setter.invoke(instance, value);
        } catch (Exception e) {
            _throwAsIOE(e, value);
        }
    }
}
```

```
/**
 * This concrete sub-class implements Collection or Map property that is
 * indirectly by getting the property value and directly modifying it.
 */
public final static class SetterlessProperty
    extends SettableBeanProperty
{
    protected final AnnotatedMethod _annotated;

    /**
     * Get method for accessing property value used to access property
     * (of Collection or Map type) to modify.
     */
    protected final Method _getter;

    public SetterlessProperty(String name, JavaType type, TypeDeserializer typeDeser,
            Annotations contextAnnotations, AnnotatedMethod method)
        {
        super(name, type, typeDeser, contextAnnotations);
        _annotated = method;
        _getter = method.getAnnotated();
    }

    /*
    /**********************************************************
    /* BeanProperty impl
    /**********************************************************
     */

    @Override
    public <A extends Annotation> A getAnnotation(Class<A> acls) {
        return _annotated.getAnnotation(acls);
    }

    @Override public AnnotatedMember getMember() {  return _annotated; }

    /*
    /**********************************************************
    /* Overridden methods
    /**********************************************************
     */

    @Override
    public final void deserializeAndSet(JsonParser jp, DeserializationContext ctxt,
            Object instance)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();
```

```
            if (t == JsonToken.VALUE_NULL) {
                /* Hmmh. Is this a problem? We won't be setting anything, so it's
                 * equivalent of empty Collection/Map in this case
                 */
                return;
            }

            // Ok: then, need to fetch Collection/Map to modify:
            Object toModify;
            try {
                toModify = _getter.invoke(instance);
            } catch (Exception e) {
                _throwAsIOE(e);
                return; // never gets here
            }
            /* Note: null won't work, since we can't then inject anything
             * in. At least that's not good in common case. However,
             * theoretically the case where we get JSON null might
             * be compatible. If so, implementation could be changed.
             */
            if (toModify == null) {
                throw new JsonMappingException("Problem deserializing 'setterless' property '"+getName()+"': get
method returned null");
            }
            _valueDeserializer.deserialize(jp, ctxt, toModify);
        }

        @Override
        public final void set(Object instance, Object value)
            throws IOException
        {
            throw new UnsupportedOperationException("Should never call 'set' on setterless property");
        }
    }

    /**
     * This concrete sub-class implements property that is set
     * directly assigning to a Field.
     */
    public final static class FieldProperty
        extends SettableBeanProperty
    {
        protected final AnnotatedField _annotated;

        /**
         * Actual field to set when deserializing this property.
         */
        protected final Field _field;
```

```
    public FieldProperty(String name, JavaType type, TypeDeserializer typeDeser,
            Annotations contextAnnotations, AnnotatedField field)
    {
        super(name, type, typeDeser, contextAnnotations);
        _annotated = field;
        _field = field.getAnnotated();
    }

    /*
    /**********************************************************
    /* BeanProperty impl
    /**********************************************************
     */

    @Override
    public <A extends Annotation> A getAnnotation(Class<A> acls) {
        return _annotated.getAnnotation(acls);
    }

    @Override public AnnotatedMember getMember() {  return _annotated; }

    /*
    /**********************************************************
    /* Overridden methods
    /**********************************************************
     */

    @Override
    public void deserializeAndSet(JsonParser jp, DeserializationContext ctxt,
                                  Object instance)
        throws IOException, JsonProcessingException
    {
        set(instance, deserialize(jp, ctxt));
    }

    @Override
    public final void set(Object instance, Object value)
        throws IOException
    {
        try {
            _field.set(instance, value);
        } catch (Exception e) {
            _throwAsIOE(e, value);
        }
    }
}
```

```
/**
 * This concrete sub-class implements property that is passed
 * via Creator (constructor or static factory method).
 */
public final static class CreatorProperty
    extends SettableBeanProperty
{
    protected final AnnotatedParameter _annotated;

    /**
     * Index of the property
     */
    final protected int _index;

    public CreatorProperty(String name, JavaType type, TypeDeserializer typeDeser,
            Annotations contextAnnotations, AnnotatedParameter param,
            int index)
    {
        super(name, type, typeDeser, contextAnnotations);
        _annotated = param;
        _index = index;
    }

    /*
    /**********************************************************
    /* BeanProperty impl
    /**********************************************************
     */

    @Override
    public <A extends Annotation> A getAnnotation(Class<A> acls) {
        return _annotated.getAnnotation(acls);
    }

    @Override public AnnotatedMember getMember() {  return _annotated; }

    /*
    /**********************************************************
    /* Overridden methods
    /**********************************************************
     */

    /**
     * Method to use for accessing index of the property (related to
     * other properties in the same context); currently only applicable
     * to "Creator properties".
     *<p>
     * Base implementation returns -1 to indicate that no index exists
```

```
     * for the property.
     */
    @Override
    public int getCreatorIndex() { return _index; }

    @Override
    public void deserializeAndSet(JsonParser jp, DeserializationContext ctxt,
                                  Object instance)
        throws IOException, JsonProcessingException
    {
        set(instance, deserialize(jp, ctxt));
    }

    @Override
    public void set(Object instance, Object value)
        throws IOException
    {
        /* Hmmmh. Should we return quietly (NOP), or error?
         * For now, let's just bail out without fuss.
         */
        //throw new IllegalStateException("Method should never be called on a "+getClass().getName());
    }
}

/**
 * Wrapper property that is used to handle managed (forward) properties
 * (see [JACKSON-235] for more information). Basically just need to
 * delegate first to actual forward property, and
 *
 * @author tatu
 */
public final static class ManagedReferenceProperty
    extends SettableBeanProperty
{
    protected final String _referenceName;

    /**
     * Flag that indicates whether property to handle is a container type
     * (array, Collection, Map) or not.
     */
    protected final boolean _isContainer;

    protected final SettableBeanProperty _managedProperty;

    protected final SettableBeanProperty _backProperty;

    public ManagedReferenceProperty(String refName,
            SettableBeanProperty forward, SettableBeanProperty backward,
```

```
        Annotations contextAnnotations,
        boolean isContainer)
{
    super(forward.getName(), forward.getType(), forward._valueTypeDeserializer,
            contextAnnotations);
    _referenceName = refName;
    _managedProperty = forward;
    _backProperty = backward;
    _isContainer = isContainer;
}

/*
/**********************************************************
/* BeanProperty impl
/**********************************************************
 */

@Override
public <A extends Annotation> A getAnnotation(Class<A> acls) {
    return _managedProperty.getAnnotation(acls);
}

@Override public AnnotatedMember getMember() {  return _managedProperty.getMember(); }

/*
/**********************************************************
/* Overridden methods
/**********************************************************
 */

@Override
public void deserializeAndSet(JsonParser jp, DeserializationContext ctxt,
                        Object instance)
    throws IOException, JsonProcessingException
{
    set(instance, _managedProperty.deserialize(jp, ctxt));
}

@Override
public final void set(Object instance, Object value)
    throws IOException
{
    _managedProperty.set(instance, value);
    /* And then back reference, if (and only if!) we actually have a non-null
     * reference
     */
    if (value != null) {
        if (_isContainer) { // ok, this gets ugly... but has to do for now
```

```
                if (value instanceof Object[]) {
                    for (Object ob : (Object[]) value) {
                        if (ob != null) {
                            _backProperty.set(ob, instance);
                        }
                    }
                } else if (value instanceof Collection<?>) {
                    for (Object ob : (Collection<?>) value) {
                        if (ob != null) {
                            _backProperty.set(ob, instance);
                        }
                    }
                } else if (value instanceof Map<?,?>) {
                    for (Object ob : ((Map<?,?>) value).values()) {
                        if (ob != null) {
                            _backProperty.set(ob, instance);
                        }
                    }
                } else {
                    throw new IllegalStateException("Unsupported container type ("+value.getClass().getName()
                            +") when resolving reference '"+_referenceName+"'");
                }
            } else {
                _backProperty.set(value, instance);
            }
        }
    }
}

/**
 * To support [JACKSON-420] we need bit more indirection; this is used to produce
 * artificial failure for primitives that don't accept JSON null as value.
 */
protected final static class NullProvider
{
    private final Object _nullValue;

    private final boolean _isPrimitive;

    private final Class<?> _rawType;

    protected NullProvider(JavaType type, Object nullValue)
    {
        _nullValue = nullValue;
        // [JACKSON-420]
        _isPrimitive = type.isPrimitive();
        _rawType = type.getRawClass();
    }
```

```
    public Object nullValue(DeserializationContext ctxt) throws JsonProcessingException
    {
        if (_isPrimitive && ctxt.isEnabled(DeserializationConfig.Feature.FAIL_ON_NULL_FOR_PRIMITIVES)) {
            throw ctxt.mappingException("Can not map JSON null into type "+_rawType.getName()
                    +" (set DeserializationConfig.Feature.FAIL_ON_NULL_FOR_PRIMITIVES to 'false' to allow)");
        }
        return _nullValue;
    }
  }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.sql.Timestamp;

import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.map.DeserializationContext;

/**
* Simple deserializer for handling {@link java.sql.Timestamp} values.
*<p>
* One way to customize Timestamp formats accepted is to override method
* {@link DeserializationContext#parseDate} that this basic
* deserializer calls.
*/
public class TimestampDeserializer
  extends StdScalarDeserializer<Timestamp>
{
  public TimestampDeserializer() { super(Timestamp.class); }

  @Override
  public java.sql.Timestamp deserialize(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
    return new Timestamp(_parseDate(jp, ctxt).getTime());
  }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.lang.reflect.Method;

import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.DeserializationConfig;
```

```
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.annotate.JsonCachable;
import org.codehaus.jackson.map.introspect.AnnotatedMethod;
import org.codehaus.jackson.map.util.ClassUtil;

/**
* Deserializer class that can deserialize instances of
* specified Enum class from Strings and Integers.
*/
@JsonCachable
/**
* Construction of these deserializers is bit costly, plus it's
* absolutely safe to cache them as well (no generic variations etc).
*/
public class EnumDeserializer
  extends StdScalarDeserializer<Enum<?>>
{
  final EnumResolver<?> _resolver;

  public EnumDeserializer(EnumResolver<?> res)
  {
     super(Enum.class);
     _resolver = res;
  }

  /**
   * Factory method used when Enum instances are to be deserialized
   * using a creator (static factory method)
   *
   * @return Deserializer based on given factory method, if type was suitable;
   *  null if type can not be used
   *
   * @since 1.6
   */
  public static JsonDeserializer<?> deserializerForCreator(DeserializationConfig config,
        Class<?> enumClass, AnnotatedMethod factory)
  {
     // note: caller has verified there's just one arg; but we must verify its type
     if (factory.getParameterType(0) != String.class) {
        throw new IllegalArgumentException("Parameter #0 type for factory method ("+factory+") not suitable, must
be java.lang.String");
     }
     if (config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS)) {
        ClassUtil.checkAndFixAccess(factory.getMember());
     }
     return new FactoryBasedDeserializer(enumClass, factory);
  }
```

```
    /*
    /**********************************************************
    /* Default JsonDeserializer implementation
    /**********************************************************
     */

    @Override
    public Enum<?> deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        JsonToken curr = jp.getCurrentToken();

        // Usually should just get string value:
        if (curr == JsonToken.VALUE_STRING) {
            String name = jp.getText();
            Enum<?> result = _resolver.findEnum(name);
            if (result == null) {
                throw ctxt.weirdStringException(_resolver.getEnumClass(), "value not one of declared Enum instance
names");
            }
            return result;
        }
        // But let's consider int acceptable as well (if within ordinal range)
        if (curr == JsonToken.VALUE_NUMBER_INT) {
            /* ... unless told not to do that. :-)
             * (as per [JACKSON-412]
             */
            if (ctxt.isEnabled(DeserializationConfig.Feature.FAIL_ON_NUMBERS_FOR_ENUMS)) {
                throw ctxt.mappingException("Not allowed to deserialize Enum value out of JSON number (disable
DeserializationConfig.Feature.FAIL_ON_NUMBERS_FOR_ENUMS to allow)");
            }

            int index = jp.getIntValue();
            Enum<?> result = _resolver.getEnum(index);
            if (result == null) {
                throw ctxt.weirdNumberException(_resolver.getEnumClass(), "index value outside legal index range
[0.."+_resolver.lastValidIndex()+"]");
            }
            return result;
        }
        throw ctxt.mappingException(_resolver.getEnumClass());
    }

    /*
    /**********************************************************
    /* Default JsonDeserializer implementation
    /**********************************************************
```

```
     */

    /**
     * Deserializer that uses a single-String static factory method
     * for locating Enum values by String id.
     */
    protected static class FactoryBasedDeserializer
        extends StdScalarDeserializer<Object>
    {
        protected final Class<?> _enumClass;
        protected final Method _factory;

        public FactoryBasedDeserializer(Class<?> cls, AnnotatedMethod f)
        {
            super(Enum.class);
            _enumClass = cls;
            _factory = f.getAnnotated();
        }

        @Override
        public Object deserialize(JsonParser jp, DeserializationContext ctxt)
            throws IOException, JsonProcessingException
        {
            JsonToken curr = jp.getCurrentToken();

            // Usually should just get string value:
            if (curr != JsonToken.VALUE_STRING) {
                throw ctxt.mappingException(_enumClass);
            }
            String value = jp.getText();
            try {
                return _factory.invoke(_enumClass, value);
            } catch (Exception e) {
                ClassUtil.unwrapAndThrowAsIAE(e);
            }
            return null;
        }
    }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import org.codehaus.jackson.JsonProcessingException;

/**
* Base class for property values that need to be buffered during
* deserialization.
*/
```

```
abstract class PropertyValue
{
  public final PropertyValue next;

  /**
   * Value to assign when POJO has been instantiated.
   */
  public final Object value;

  protected PropertyValue(PropertyValue next, Object value)
  {
     this.next = next;
     this.value = value;
  }

  /**
   * Method called to assign stored value of this property to specified
   * bean instance
   */
  public abstract void assign(Object bean)
     throws IOException, JsonProcessingException;

  /*
  /**********************************************************
  /* Concrete property value classes
  /**********************************************************
   */

  /**
   * Property value that used when assigning value to property using
   * a setter method or direct field access.
   */
  final static class Regular
     extends PropertyValue
  {
     final SettableBeanProperty _property;

     public Regular(PropertyValue next, Object value,
                    SettableBeanProperty prop)
     {
        super(next, value);
        _property = prop;
     }

     @Override
     public void assign(Object bean)
        throws IOException, JsonProcessingException
     {
```

```
        _property.set(bean, value);
    }
}

/**
 * Property value type used when storing entries to be added
 * to a POJO using "any setter" (method that takes name and
 * value arguments, allowing setting multiple different
 * properties using single method).
 */
final static class Any
    extends PropertyValue
{
    final SettableAnyProperty _property;
    final String _propertyName;

    public Any(PropertyValue next, Object value,
            SettableAnyProperty prop,
            String propName)
    {
        super(next, value);
        _property = prop;
        _propertyName = propName;
    }

    @Override
    public void assign(Object bean)
        throws IOException, JsonProcessingException
    {
        _property.set(bean, _propertyName, value);
    }
}

/**
 * Property value type used when storing entries to be added
 * to a Map.
 */
final static class Map
    extends PropertyValue
{
    final Object _key;

    public Map(PropertyValue next, Object value, Object key)
    {
        super(next, value);
        _key = key;
    }
```

```
        @SuppressWarnings("unchecked")
        @Override
        public void assign(Object bean)
            throws IOException, JsonProcessingException
        {
            ((java.util.Map<Object,Object>) bean).put(_key, value);
        }
    }
}
package org.codehaus.jackson.map.deser;

import java.util.HashMap;
import java.util.HashSet;

import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.deser.impl.BeanPropertyMap;
import org.codehaus.jackson.map.introspect.BasicBeanDescription;

/**
* Builder class used for aggregating deserialization information about
* a POJO, in order to build a {@link JsonDeserializer} for deserializing
* intances.
*
* @since 1.7
*/
public class BeanDeserializerBuilder
{
    /*
    /**********************************************************
    /* General information about POJO
    /**********************************************************
     */

    final protected BasicBeanDescription _beanDesc;

    /*
    /**********************************************************
    /* Accumulated information about properties
    /**********************************************************
     */

    /**
     * Properties to deserialize collected so far.
     */
    final protected HashMap<String, SettableBeanProperty> _properties = new HashMap<String,
SettableBeanProperty>();

    /**
```

```
     * Back-reference properties this bean contains (if any)
     */
    protected HashMap<String, SettableBeanProperty> _backRefProperties;

    /**
     * Set of names of properties that are recognized but are to be ignored for deserialization
     * purposes (meaning no exception is thrown, value is just skipped).
     */
    protected HashSet<String> _ignorableProps;

    /**
     * Set of creators (constructors, factory methods) that
     * bean type has.
     */
    protected CreatorContainer _creators;

    /**
     * Fallback setter used for handling any properties that are not
     * mapped to regular setters. If setter is not null, it will be
     * called once for each such property.
     */
    protected SettableAnyProperty _anySetter;

    /**
     * Flag that can be set to ignore and skip unknown properties.
     * If set, will not throw an exception for unknown properties.
     */
    protected boolean _ignoreAllUnknown;

    /*
    /**********************************************************
    /* Construction and setters
    /**********************************************************
     */

    public BeanDeserializerBuilder(BasicBeanDescription beanDesc)
    {
        _beanDesc = beanDesc;
    }

    public void setCreators(CreatorContainer creators) {
        _creators = creators;
    }

    /**
     * Method for adding a new property or replacing a property.
     */
    public void addOrReplaceProperty(SettableBeanProperty prop, boolean allowOverride)
```

```
    {
        _properties.put(prop.getName(), prop);
    }

    /**
     * Method to add a property setter. Will ensure that there is no
     * unexpected override; if one is found will throw a
     * {@link IllegalArgumentException}.
     */
    public void addProperty(SettableBeanProperty prop)
    {
        SettableBeanProperty old =  _properties.put(prop.getName(), prop);
        if (old != null && old != prop) { // should never occur...
            throw new IllegalArgumentException("Duplicate property '"+prop.getName()+"' for
"+_beanDesc.getType());
        }
    }

    public void  addBackReferenceProperty(String referenceName, SettableBeanProperty prop)
    {
        if (_backRefProperties == null) {
            _backRefProperties = new HashMap<String, SettableBeanProperty>(4);
        }
        _backRefProperties.put(referenceName, prop);
    }

    /**
     * Method that will add property name as one of properties that can
     * be ignored if not recognized.
     */
    public void addIgnorable(String propName)
    {
        if (_ignorableProps == null) {
            _ignorableProps = new HashSet<String>();
        }
        _ignorableProps.add(propName);
    }

    public boolean hasProperty(String propertyName) {
        return _properties.containsKey(propertyName);
    }

    public SettableBeanProperty removeProperty(String name)
    {
        return _properties.remove(name);
    }

    public void setAnySetter(SettableAnyProperty s)
```

```
    {
        if (_anySetter != null && s != null) {
            throw new IllegalStateException("_anySetter already set to non-null");
        }
        _anySetter = s;
    }

    public void setIgnoreUnknownProperties(boolean ignore) {
        _ignoreAllUnknown = ignore;
    }

    /*
    /**********************************************************
    /* Build method(s)
    /**********************************************************
     */

    public JsonDeserializer<?> build(BeanProperty forProperty)
    {
        BeanPropertyMap propertyMap = new BeanPropertyMap(_properties.values());
        propertyMap.assignIndexes();

        return new BeanDeserializer(_beanDesc.getClassInfo(), _beanDesc.getType(), forProperty,
                _creators, propertyMap, _backRefProperties, _ignorableProps, _ignoreAllUnknown,
                _anySetter);
    }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.*;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicReference;

import org.codehaus.jackson.Base64Variants;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.io.NumberInput;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.util.TokenBuffer;

/**
```

```
* Base class for common deserializers. Contains shared
* base functionality for dealing with primitive values, such
* as (re)parsing from String.
*/
public abstract class StdDeserializer<T>
  extends JsonDeserializer<T>
{
  /**
   * Type of values this deserializer handles: sometimes
   * exact types, other time most specific supertype of
   * types deserializer handles (which may be as generic
   * as {@link Object} in some case)
   */
  final protected Class<?> _valueClass;

  protected StdDeserializer(Class<?> vc) {
     _valueClass = vc;
  }

  /**
   * @since 1.7
   */
  protected StdDeserializer(JavaType valueType) {
     _valueClass = (valueType == null) ? null : valueType.getRawClass();
  }

  /*
  /**********************************************************
  /* Extended API
  /**********************************************************
   */

  public Class<?> getValueClass() { return _valueClass; }

  /**
   * Exact structured type deserializer handles, if known.
   *<p>
   * Default implementation just returns null.
   */
  public JavaType getValueType() { return null; }

  /**
   * Method that can be called to determine if given deserializer is the default
   * deserializer Jackson uses; as opposed to a custom deserializer installed by
   * a module or calling application. Determination is done using
   * {@link JacksonStdImpl} annotation on deserializer class.
   *
   * @since 1.7
```

```
 */
protected boolean isDefaultSerializer(JsonDeserializer<?> deserializer)
{
   return (deserializer != null && deserializer.getClass().getAnnotation(JacksonStdImpl.class) != null);
}

/*
/**********************************************************
/* Partial JsonDeserializer implementation
/**********************************************************
 */

/**
 * Base implementation that does not assume specific type
 * inclusion mechanism. Sub-classes are expected to override
 * this method if they are to handle type information.
 */
@Override
public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
      TypeDeserializer typeDeserializer)
   throws IOException, JsonProcessingException
{
   return typeDeserializer.deserializeTypedFromAny(jp, ctxt);
}

/*
/**********************************************************
/* Helper methods for sub-classes, parsing
/**********************************************************
 */

protected final boolean _parseBooleanPrimitive(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   JsonToken t = jp.getCurrentToken();
   if (t == JsonToken.VALUE_TRUE) {
      return true;
   }
   if (t == JsonToken.VALUE_FALSE) {
      return false;
   }
   if (t == JsonToken.VALUE_NULL) {
      return false;
   }
   // [JACKSON-78]: should accept ints too, (0 == false, otherwise true)
   if (t == JsonToken.VALUE_NUMBER_INT) {
      return (jp.getIntValue() != 0);
   }
```

```
    // And finally, let's allow Strings to be converted too
    if (t == JsonToken.VALUE_STRING) {
        String text = jp.getText().trim();
        if ("true".equals(text)) {
            return true;
        }
        if ("false".equals(text) || text.length() == 0) {
            return Boolean.FALSE;
        }
        throw ctxt.weirdStringException(_valueClass, "only \"true\" or \"false\" recognized");
    }
    // Otherwise, no can do:
    throw ctxt.mappingException(_valueClass);
}

protected final Boolean _parseBoolean(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    JsonToken t = jp.getCurrentToken();
    if (t == JsonToken.VALUE_TRUE) {
        return Boolean.TRUE;
    }
    if (t == JsonToken.VALUE_FALSE) {
        return Boolean.FALSE;
    }
    if (t == JsonToken.VALUE_NULL) {
        return null;
    }
    // [JACKSON-78]: should accept ints too, (0 == false, otherwise true)
    if (t == JsonToken.VALUE_NUMBER_INT) {
        return (jp.getIntValue() == 0) ? Boolean.FALSE : Boolean.TRUE;
    }
    // And finally, let's allow Strings to be converted too
    if (t == JsonToken.VALUE_STRING) {
        String text = jp.getText().trim();
        if ("true".equals(text)) {
            return Boolean.TRUE;
        }
        if ("false".equals(text) || text.length() == 0) {
            return Boolean.FALSE;
        }
        throw ctxt.weirdStringException(_valueClass, "only \"true\" or \"false\" recognized");
    }
    // Otherwise, no can do:
    throw ctxt.mappingException(_valueClass);
}

protected final Short _parseShort(JsonParser jp, DeserializationContext ctxt)
```

```
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();
        if (t == JsonToken.VALUE_NULL) {
            return null;
        }
        if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) { // coercing
should work too
            return jp.getShortValue();
        }
        int value = _parseIntPrimitive(jp, ctxt);
        // So far so good: but does it fit?
        if (value < Short.MIN_VALUE || value > Short.MAX_VALUE) {
            throw ctxt.weirdStringException(_valueClass, "overflow, value can not be represented as 16-bit value");
        }
        return (short) value;
    }

    protected final short _parseShortPrimitive(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        int value = _parseIntPrimitive(jp, ctxt);
        // So far so good: but does it fit?
        if (value < Short.MIN_VALUE || value > Short.MAX_VALUE) {
            throw ctxt.weirdStringException(_valueClass, "overflow, value can not be represented as 16-bit value");
        }
        return (short) value;
    }

    protected final int _parseIntPrimitive(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();

        // Int works as is, coercing fine as well
        if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) { // coercing
should work too
            return jp.getIntValue();
        }
        if (t == JsonToken.VALUE_STRING) { // let's do implicit re-parse
            /* 31-Dec-2009, tatus: Should improve handling of overflow
             *   values... but this'll have to do for now
             */
            String text = jp.getText().trim();
            try {
                int len = text.length();
                if (len > 9) {
                    long l = Long.parseLong(text);
```

```
                if (l < Integer.MIN_VALUE || l > Integer.MAX_VALUE) {
                    throw ctxt.weirdStringException(_valueClass,
                        "Overflow: numeric value ("+text+") out of range of int ("+Integer.MIN_VALUE+" -
"+Integer.MAX_VALUE+")");
                }
                return (int) l;
            }
            if (len == 0) {
                return 0;
            }
            return NumberInput.parseInt(text);
        } catch (IllegalArgumentException iae) {
            throw ctxt.weirdStringException(_valueClass, "not a valid int value");
        }
    }
    if (t == JsonToken.VALUE_NULL) {
        return 0;
    }
    // Otherwise, no can do:
    throw ctxt.mappingException(_valueClass);
  }

  protected final Integer _parseInteger(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
    JsonToken t = jp.getCurrentToken();
    if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) { // coercing
should work too
        return Integer.valueOf(jp.getIntValue());
    }
    if (t == JsonToken.VALUE_STRING) { // let's do implicit re-parse
        String text = jp.getText().trim();
        try {
            int len = text.length();
            if (len > 9) {
                long l = Long.parseLong(text);
                if (l < Integer.MIN_VALUE || l > Integer.MAX_VALUE) {
                    throw ctxt.weirdStringException(_valueClass,
                        "Overflow: numeric value ("+text+") out of range of Integer ("+Integer.MIN_VALUE+" -
"+Integer.MAX_VALUE+")");
                }
                return Integer.valueOf((int) l);
            }
            if (len == 0) {
                return null;
            }
            return Integer.valueOf(NumberInput.parseInt(text));
        } catch (IllegalArgumentException iae) {
```

```
            throw ctxt.weirdStringException(_valueClass, "not a valid Integer value");
        }
    }
    if (t == JsonToken.VALUE_NULL) {
        return null;
    }
    // Otherwise, no can do:
    throw ctxt.mappingException(_valueClass);
}

protected final Long _parseLong(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    JsonToken t = jp.getCurrentToken();

    // it should be ok to coerce (although may fail, too)
    if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) {
        return jp.getLongValue();
    }
    // let's allow Strings to be converted too
    if (t == JsonToken.VALUE_STRING) {
        // !!! 05-Jan-2009, tatu: Should we try to limit value space, JDK is too lenient?
        String text = jp.getText().trim();
        if (text.length() == 0) {
            return null;
        }
        try {
            return Long.valueOf(NumberInput.parseLong(text));
        } catch (IllegalArgumentException iae) { }
        throw ctxt.weirdStringException(_valueClass, "not a valid Long value");
    }
    if (t == JsonToken.VALUE_NULL) {
        return null;
    }
    // Otherwise, no can do:
    throw ctxt.mappingException(_valueClass);
}

protected final long _parseLongPrimitive(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    JsonToken t = jp.getCurrentToken();
    if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) {
        return jp.getLongValue();
    }
    if (t == JsonToken.VALUE_STRING) {
        String text = jp.getText().trim();
        if (text.length() == 0) {
```

```
            return 0L;
        }
        try {
            return NumberInput.parseLong(text);
        } catch (IllegalArgumentException iae) { }
        throw ctxt.weirdStringException(_valueClass, "not a valid long value");
    }
    if (t == JsonToken.VALUE_NULL) {
        return 0L;
    }
    throw ctxt.mappingException(_valueClass);
}

protected final Float _parseFloat(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    // We accept couple of different types; obvious ones first:
    JsonToken t = jp.getCurrentToken();

    if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) { // coercing
should work too
        return jp.getFloatValue();
    }
    // And finally, let's allow Strings to be converted too
    if (t == JsonToken.VALUE_STRING) {
        String text = jp.getText().trim();
        if (text.length() == 0) {
            return null;
        }
        switch (text.charAt(0)) {
        case 'I':
            if ("Infinity".equals(text) || "INF".equals(text)) {
                return Float.POSITIVE_INFINITY;
            }
            break;
        case 'N':
            if ("NaN".equals(text)) {
                return Float.NaN;
            }
            break;
        case '-':
            if ("-Infinity".equals(text) || "-INF".equals(text)) {
                return Float.NEGATIVE_INFINITY;
            }
            break;
        }
        try {
            return Float.parseFloat(text);
```

```
            } catch (IllegalArgumentException iae) { }
            throw ctxt.weirdStringException(_valueClass, "not a valid Float value");
        }
        if (t == JsonToken.VALUE_NULL) {
            return null;
        }
        // Otherwise, no can do:
        throw ctxt.mappingException(_valueClass);
    }

    protected final float _parseFloatPrimitive(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();

        if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) { // coercing
should work too
            return jp.getFloatValue();
        }
        if (t == JsonToken.VALUE_STRING) {
            String text = jp.getText().trim();
            if (text.length() == 0) {
                return 0.0f;
            }
            switch (text.charAt(0)) {
            case 'I':
                if ("Infinity".equals(text) || "INF".equals(text)) {
                    return Float.POSITIVE_INFINITY;
                }
                break;
            case 'N':
                if ("NaN".equals(text)) {
                    return Float.NaN;
                }
                break;
            case '-':
                if ("-Infinity".equals(text) || "-INF".equals(text)) {
                    return Float.NEGATIVE_INFINITY;
                }
                break;
            }
            try {
                return Float.parseFloat(text);
            } catch (IllegalArgumentException iae) { }
            throw ctxt.weirdStringException(_valueClass, "not a valid float value");
        }
        if (t == JsonToken.VALUE_NULL) {
            return 0.0f;
```

```
        }
        // Otherwise, no can do:
        throw ctxt.mappingException(_valueClass);
    }

    protected final Double _parseDouble(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();

        if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) { // coercing
should work too
            return jp.getDoubleValue();
        }
        if (t == JsonToken.VALUE_STRING) {
            String text = jp.getText().trim();
            if (text.length() == 0) {
                return null;
            }
            switch (text.charAt(0)) {
            case 'I':
                if ("Infinity".equals(text) || "INF".equals(text)) {
                    return Double.POSITIVE_INFINITY;
                }
                break;
            case 'N':
                if ("NaN".equals(text)) {
                    return Double.NaN;
                }
                break;
            case '-':
                if ("-Infinity".equals(text) || "-INF".equals(text)) {
                    return Double.NEGATIVE_INFINITY;
                }
                break;
            }
            try {
                return parseDouble(text);
            } catch (IllegalArgumentException iae) { }
            throw ctxt.weirdStringException(_valueClass, "not a valid Double value");
        }
        if (t == JsonToken.VALUE_NULL) {
            return null;
        }
            // Otherwise, no can do:
        throw ctxt.mappingException(_valueClass);
    }
```

```
  protected final double _parseDoublePrimitive(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
    // We accept couple of different types; obvious ones first:
    JsonToken t = jp.getCurrentToken();

    if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) { // coercing
should work too
      return jp.getDoubleValue();
    }
    // And finally, let's allow Strings to be converted too
    if (t == JsonToken.VALUE_STRING) {
      String text = jp.getText().trim();
      if (text.length() == 0) {
        return 0.0;
      }
      switch (text.charAt(0)) {
      case 'I':
        if ("Infinity".equals(text) || "INF".equals(text)) {
          return Double.POSITIVE_INFINITY;
        }
        break;
      case 'N':
        if ("NaN".equals(text)) {
          return Double.NaN;
        }
        break;
      case '-':
        if ("-Infinity".equals(text) || "-INF".equals(text)) {
          return Double.NEGATIVE_INFINITY;
        }
        break;
      }
      try {
        return parseDouble(text);
      } catch (IllegalArgumentException iae) { }
      throw ctxt.weirdStringException(_valueClass, "not a valid double value");
    }
    if (t == JsonToken.VALUE_NULL) {
      return 0.0;
    }
      // Otherwise, no can do:
    throw ctxt.mappingException(_valueClass);
  }


  protected java.util.Date _parseDate(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
```

```
    {
        JsonToken t = jp.getCurrentToken();
        try {
            if (t == JsonToken.VALUE_NUMBER_INT) {
                return new java.util.Date(jp.getLongValue());
            }
            if (t == JsonToken.VALUE_STRING) {
                /* As per [JACKSON-203], take empty Strings to mean
                 * null
                 */
                String str = jp.getText().trim();
                if (str.length() == 0) {
                    return null;
                }
                return ctxt.parseDate(str);
            }
            throw ctxt.mappingException(_valueClass);
        } catch (IllegalArgumentException iae) {
            throw ctxt.weirdStringException(_valueClass, "not a valid representation (error: "+iae.getMessage()+")");
        }
    }

    /**
     * Helper method for encapsulating calls to low-level double value parsing; single place
     * just because we need a work-around that must be applied to all calls.
     *<p>
     * Note: copied from <code>org.codehaus.jackson.io.NumberUtil</code> (to avoid dependency to
     * version 1.8; except for String constants, but that gets compiled in bytecode here)
     */
    protected final static double parseDouble(String numStr) throws NumberFormatException
    {
        // [JACKSON-486]: avoid some nasty float representations... but should it be MIN_NORMAL or
MIN_VALUE?
        if (NumberInput.NASTY_SMALL_DOUBLE.equals(numStr)) {
            return Double.MIN_NORMAL;
        }
        return Double.parseDouble(numStr);
    }

    /*
    /**********************************************************
    /* Helper methods for sub-classes, resolving dependencies
    /**********************************************************
    */

    /**
     * Helper method used to locate deserializers for properties the
     * type this deserializer handles contains (usually for properties of
```

```
     * bean types)
     *
     * @param config Active deserialization configuration
     * @param provider Deserializer provider to use for actually finding deserializer(s)
     * @param type Type of property to deserialize
     * @param property Actual property object (field, method, constuctor parameter) used
     *    for passing deserialized values; provided so deserializer can be contextualized if necessary (since 1.7)
     */
    protected JsonDeserializer<Object> findDeserializer(DeserializationConfig config, DeserializerProvider provider,
                                        JavaType type, BeanProperty property)
        throws JsonMappingException
    {
        JsonDeserializer<Object> deser = provider.findValueDeserializer(config, type, property);
        return deser;
    }

    /*
    /**********************************************************
    /* Helper methods for sub-classes, problem reporting
    /**********************************************************
     */

    /**
     * Method called to deal with a property that did not map to a known
     * Bean property. Method can deal with the problem as it sees fit (ignore,
     * throw exception); but if it does return, it has to skip the matching
     * Json content parser has.
     *<p>
     * NOTE: method signature was changed in version 1.5; explicit JsonParser
     * <b>must</b> be passed since it may be something other than what
     * context has. Prior versions did not include the first parameter.
     *
     * @param jp Parser that points to value of the unknown property
     * @param ctxt Context for deserialization; allows access to the parser,
     *    error reporting functionality
     * @param instanceOrClass Instance that is being populated by this
     *   deserializer, or if not known, Class that would be instantiated.
     *   If null, will assume type is what {@link #getValueClass} returns.
     * @param propName Name of the property that can not be mapped
     */
    protected void handleUnknownProperty(JsonParser jp, DeserializationContext ctxt, Object instanceOrClass, String
propName)
        throws IOException, JsonProcessingException
    {
        if (instanceOrClass == null) {
            instanceOrClass = getValueClass();
        }
        // Maybe we have configured handler(s) to take care of it?
```

```
    if (ctxt.handleUnknownProperty(jp, this, instanceOrClass, propName)) {
        return;
    }
    // Nope, not handled. Potentially that's a problem...
    reportUnknownProperty(ctxt, instanceOrClass, propName);

    /* If we get this far, need to skip now; we point to first token of
     * value (START_xxx for structured, or the value token for others)
     */
    jp.skipChildren();
}

protected void reportUnknownProperty(DeserializationContext ctxt,
                             Object instanceOrClass, String fieldName)
    throws IOException, JsonProcessingException
{
    // throw exception if that's what we are expected to do
    if (ctxt.isEnabled(DeserializationConfig.Feature.FAIL_ON_UNKNOWN_PROPERTIES)) {
        throw ctxt.unknownFieldException(instanceOrClass, fieldName);
    }
    // ... or if not, just ignore
}


/*
/**********************************************************
/* Then one intermediate base class for things that have
/* both primitive and wrapper types
/**********************************************************
 */

protected abstract static class PrimitiveOrWrapperDeserializer<T>
    extends StdScalarDeserializer<T>
{
    final T _nullValue;

    protected PrimitiveOrWrapperDeserializer(Class<T> vc, T nvl)
    {
        super(vc);
        _nullValue = nvl;
    }

    @Override
    public final T getNullValue() {
        return _nullValue;
    }
}
```

```
/*
/**********************************************************
/* First, generic (Object, String, String-like, Class) deserializers
/**********************************************************
 */

@JacksonStdImpl
public final static class StringDeserializer
   extends StdScalarDeserializer<String>
{
   public StringDeserializer() { super(String.class); }

   @Override
   public String deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException
   {
      JsonToken curr = jp.getCurrentToken();
      // Usually should just get string value:
      if (curr == JsonToken.VALUE_STRING) {
         return jp.getText();
      }
      // [JACKSON-330]: need to gracefully handle byte[] data, as base64
      if (curr == JsonToken.VALUE_EMBEDDED_OBJECT) {
         Object ob = jp.getEmbeddedObject();
         if (ob == null) {
            return null;
         }
         if (ob instanceof byte[]) {
            return Base64Variants.getDefaultVariant().encode((byte[]) ob, false);
         }
         // otherwise, try conversion using toString()...
         return ob.toString();
      }
      // Can deserialize any scalar value, but not markers
      if (curr.isScalarValue()) {
         return jp.getText();
      }
      throw ctxt.mappingException(_valueClass);
   }

   // 1.6: since we can never have type info ("natural type"; String, Boolean, Integer, Double):
   // (is it an error to even call this version?)
   @Override
   public String deserializeWithType(JsonParser jp, DeserializationContext ctxt,
         TypeDeserializer typeDeserializer)
      throws IOException, JsonProcessingException
   {
      return deserialize(jp, ctxt);
```

```
    }
  }

  @JacksonStdImpl
  public final static class ClassDeserializer
    extends StdScalarDeserializer<Class<?>>
  {
    public ClassDeserializer() { super(Class.class); }

    @Override
      public Class<?> deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException
    {
      JsonToken curr = jp.getCurrentToken();
      // Currently will only accept if given simple class name
      if (curr == JsonToken.VALUE_STRING) {
        try {
          return Class.forName(jp.getText());
        } catch (ClassNotFoundException e) {
          throw ctxt.instantiationException(_valueClass, e);
        }
      }
      throw ctxt.mappingException(_valueClass);
    }
  }

  /*
  /**********************************************************
  /* Then primitive/wrapper types
  /**********************************************************
   */

  @JacksonStdImpl
  public final static class BooleanDeserializer
    extends PrimitiveOrWrapperDeserializer<Boolean>
  {
    public BooleanDeserializer(Class<Boolean> cls, Boolean nvl)
    {
      super(cls, nvl);
    }

    @Override
  public Boolean deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException
    {
      return _parseBoolean(jp, ctxt);
    }
```

```
    // 1.6: since we can never have type info ("natural type"; String, Boolean, Integer, Double):
    // (is it an error to even call this version?)
    @Override
    public Boolean deserializeWithType(JsonParser jp, DeserializationContext ctxt,
            TypeDeserializer typeDeserializer)
        throws IOException, JsonProcessingException
    {
        return _parseBoolean(jp, ctxt);
    }
}

@JacksonStdImpl
public final static class ByteDeserializer
    extends PrimitiveOrWrapperDeserializer<Byte>
{
    public ByteDeserializer(Class<Byte> cls, Byte nvl)
    {
        super(cls, nvl);
    }

    @Override
    public Byte deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        int value = _parseIntPrimitive(jp, ctxt);
        // So far so good: but does it fit?
        if (value < Byte.MIN_VALUE || value > Byte.MAX_VALUE) {
            throw ctxt.weirdStringException(_valueClass, "overflow, value can not be represented as 8-bit value");
        }
        return Byte.valueOf((byte) value);
    }
}

@JacksonStdImpl
public final static class ShortDeserializer
    extends PrimitiveOrWrapperDeserializer<Short>
{
    public ShortDeserializer(Class<Short> cls, Short nvl)
    {
        super(cls, nvl);
    }

    @Override
    public Short deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        return _parseShort(jp, ctxt);
    }
```

```
}

@JacksonStdImpl
public final static class CharacterDeserializer
   extends PrimitiveOrWrapperDeserializer<Character>
{
   public CharacterDeserializer(Class<Character> cls, Character nvl)
   {
      super(cls, nvl);
   }

   @Override
   public Character deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException
   {
      JsonToken t = jp.getCurrentToken();
      int value;

      if (t == JsonToken.VALUE_NUMBER_INT) { // ok iff ascii value
         value = jp.getIntValue();
         if (value >= 0 && value <= 0xFFFF) {
            return Character.valueOf((char) value);
         }
      } else if (t == JsonToken.VALUE_STRING) { // this is the usual type
         // But does it have to be exactly one char?
         String text = jp.getText();
         if (text.length() == 1) {
            return Character.valueOf(text.charAt(0));
         }
      }
      throw ctxt.mappingException(_valueClass);
   }
}

@JacksonStdImpl
public final static class IntegerDeserializer
   extends PrimitiveOrWrapperDeserializer<Integer>
{
   public IntegerDeserializer(Class<Integer> cls, Integer nvl)
   {
      super(cls, nvl);
   }

   @Override
   public Integer deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException
   {
      return _parseInteger(jp, ctxt);
```

```
    }

    // 1.6: since we can never have type info ("natural type"; String, Boolean, Integer, Double):
    // (is it an error to even call this version?)
    @Override
    public Integer deserializeWithType(JsonParser jp, DeserializationContext ctxt,
            TypeDeserializer typeDeserializer)
        throws IOException, JsonProcessingException
    {
        return _parseInteger(jp, ctxt);
    }
}

@JacksonStdImpl
public final static class LongDeserializer
    extends PrimitiveOrWrapperDeserializer<Long>
{
    public LongDeserializer(Class<Long> cls, Long nvl)
    {
        super(cls, nvl);
    }

    @Override
    public Long deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        return _parseLong(jp, ctxt);
    }
}

@JacksonStdImpl
public final static class FloatDeserializer
    extends PrimitiveOrWrapperDeserializer<Float>
{
    public FloatDeserializer(Class<Float> cls, Float nvl)
    {
        super(cls, nvl);
    }

    @Override
    public Float deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        /* 22-Jan-2009, tatu: Bounds/range checks would be tricky
         *   here, so let's not bother even trying...
         */
        return _parseFloat(jp, ctxt);
    }
```

```
}

@JacksonStdImpl
public final static class DoubleDeserializer
   extends PrimitiveOrWrapperDeserializer<Double>
{
   public DoubleDeserializer(Class<Double> cls, Double nvl)
   {
      super(cls, nvl);
   }

   @Override
   public Double deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException
   {
      return _parseDouble(jp, ctxt);
   }

   // 1.6: since we can never have type info ("natural type"; String, Boolean, Integer, Double):
   // (is it an error to even call this version?)
   @Override
   public Double deserializeWithType(JsonParser jp, DeserializationContext ctxt,
         TypeDeserializer typeDeserializer)
      throws IOException, JsonProcessingException
   {
      return _parseDouble(jp, ctxt);
   }
}

/**
 * For type <code>Number.class</code>, we can just rely on type
 * mappings that plain {@link JsonParser#getNumberValue} returns.
 *<p>
 * Since 1.5, there is one additional complication: some numeric
 * types (specifically, int/Integer and double/Double) are "non-typed";
 * meaning that they will NEVER be output with type information.
 * But other numeric types may need such type information.
 * This is why {@link #deserializeWithType} must be overridden.
 */
@JacksonStdImpl
public final static class NumberDeserializer
   extends StdScalarDeserializer<Number>
{
   public NumberDeserializer() { super(Number.class); }

   @Override
   public Number deserialize(JsonParser jp, DeserializationContext ctxt)
      throws IOException, JsonProcessingException
```

```
{
    JsonToken t = jp.getCurrentToken();
    if (t == JsonToken.VALUE_NUMBER_INT) {
        if (ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_INTEGER_FOR_INTS)) {
            return jp.getBigIntegerValue();
        }
        return jp.getNumberValue();
    } else if (t == JsonToken.VALUE_NUMBER_FLOAT) {
        /* [JACKSON-72]: need to allow overriding the behavior
         * regarding which type to use
         */
        if (ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_DECIMAL_FOR_FLOATS)) {
            return jp.getDecimalValue();
        }
        return Double.valueOf(jp.getDoubleValue());
    }

    /* Textual values are more difficult... not parsing itself, but figuring
     * out 'minimal' type to use
     */
    if (t == JsonToken.VALUE_STRING) { // let's do implicit re-parse
        String text = jp.getText().trim();
        try {
            if (text.indexOf('.') >= 0) { // floating point
                // as per [JACKSON-72]:
                if (ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_DECIMAL_FOR_FLOATS)) {
                    return new BigDecimal(text);
                }
                return new Double(text);
            }
            // as per [JACKSON-100]:
            if (ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_INTEGER_FOR_INTS)) {
                return new BigInteger(text);
            }
            long value = Long.parseLong(text);
            if (value <= Integer.MAX_VALUE && value >= Integer.MIN_VALUE) {
                return Integer.valueOf((int) value);
            }
            return Long.valueOf(value);
        } catch (IllegalArgumentException iae) {
            throw ctxt.weirdStringException(_valueClass, "not a valid number");
        }
    }
    // Otherwise, no can do:
    throw ctxt.mappingException(_valueClass);
}

/**
```

```
     * As mentioned in class Javadoc, there is additional complexity in
     * handling potentially mixed type information here. Because of this,
     * we must actually check for "raw" integers and doubles first, before
     * calling type deserializer.
     */
    @Override
    public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
                                      TypeDeserializer typeDeserializer)
        throws IOException, JsonProcessingException
    {
        switch (jp.getCurrentToken()) {
        case VALUE_NUMBER_INT:
        case VALUE_NUMBER_FLOAT:
        case VALUE_STRING:
            // can not point to type information: hence must be non-typed (int/double)
            return deserialize(jp, ctxt);
        }
        return typeDeserializer.deserializeTypedFromScalar(jp, ctxt);
    }
}

/*
/**********************************************************
/* Atomic types (java.util.concurrent.atomic), [JACKSON-283]
/*
/* note: AtomicInteger and AtomicLong work out of the box,
/* due to single-arg constructors
/**********************************************************
 */

public final static class AtomicBooleanDeserializer
    extends StdScalarDeserializer<AtomicBoolean>
{
    public AtomicBooleanDeserializer() { super(AtomicBoolean.class); }

    @Override
    public AtomicBoolean deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        // 16-Dec-2010, tatu: Should we actually convert null to null AtomicBoolean?
        return new AtomicBoolean(_parseBooleanPrimitive(jp, ctxt));
    }
}

public static class AtomicReferenceDeserializer
    extends StdScalarDeserializer<AtomicReference<?>>
    implements ResolvableDeserializer
{
```

```
    /**
     * Type of value that we reference
     */
    protected final JavaType _referencedType;

    protected final BeanProperty _property;

    protected JsonDeserializer<?> _valueDeserializer;

    /**
     * @param type AtomicReference deserializer is to be constructed for
     */
    public AtomicReferenceDeserializer(JavaType type, BeanProperty property)
    {
        super(type.getRawClass());
        JavaType[] refTypes = TypeFactory.findParameterTypes(type, AtomicReference.class);
        if (refTypes == null) { // untyped (raw)
            _referencedType = TypeFactory.type(Object.class);
        } else {
            _referencedType = refTypes[0];
        }
        _property = property;
    }

    @Override
    public AtomicReference<?> deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        return new AtomicReference<Object>(_valueDeserializer.deserialize(jp, ctxt));
    }

    @Override
    public void resolve(DeserializationConfig config, DeserializerProvider provider)
        throws JsonMappingException
    {
        _valueDeserializer = provider.findValueDeserializer(config, _referencedType, _property);
    }
}

/*
/**********************************************************
/* And then bit more complicated (but non-structured) number
/* types
/**********************************************************
 */

@JacksonStdImpl
public static class BigDecimalDeserializer
```

```
    extends StdScalarDeserializer<BigDecimal>
{
    public BigDecimalDeserializer() { super(BigDecimal.class); }

    @Override
public BigDecimal deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();
        if (t == JsonToken.VALUE_NUMBER_INT || t == JsonToken.VALUE_NUMBER_FLOAT) {
            return jp.getDecimalValue();
        }
        // String is ok too, can easily convert
        if (t == JsonToken.VALUE_STRING) { // let's do implicit re-parse
            String text = jp.getText().trim();
            if (text.length() == 0) {
                return null;
            }
            try {
                return new BigDecimal(text);
            } catch (IllegalArgumentException iae) {
                throw ctxt.weirdStringException(_valueClass, "not a valid representation");
            }
        }
        // Otherwise, no can do:
        throw ctxt.mappingException(_valueClass);
    }
}

/**
 * This is bit trickier to implement efficiently, while avoiding
 * overflow problems.
 */
@JacksonStdImpl
public static class BigIntegerDeserializer
    extends StdScalarDeserializer<BigInteger>
{
    public BigIntegerDeserializer() { super(BigInteger.class); }

    @Override
public BigInteger deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();
        String text;

        if (t == JsonToken.VALUE_NUMBER_INT) {
            switch (jp.getNumberType()) {
```

```
            case INT:
            case LONG:
                return BigInteger.valueOf(jp.getLongValue());
            }
        } else if (t == JsonToken.VALUE_NUMBER_FLOAT) {
            /* Whether to fail if there's non-integer part?
             * Could do by calling BigDecimal.toBigIntegerExact()
             */
            return jp.getDecimalValue().toBigInteger();
        } else if (t != JsonToken.VALUE_STRING) { // let's do implicit re-parse
            // String is ok too, can easily convert; otherwise, no can do:
            throw ctxt.mappingException(_valueClass);
        }
        text = jp.getText().trim();
        if (text.length() == 0) {
            return null;
        }
        try {
            return new BigInteger(text);
        } catch (IllegalArgumentException iae) {
            throw ctxt.weirdStringException(_valueClass, "not a valid representation");
        }
    }
}

/*
/**********************************************************
/* Then trickier things: Date/Calendar types
/**********************************************************
*/

@JacksonStdImpl
public static class CalendarDeserializer
    extends StdScalarDeserializer<Calendar>
{
    /**
     * We may know actual expected type; if so, it will be
     * used for instantiation.
     */
    Class<? extends Calendar> _calendarClass;

    public CalendarDeserializer() { this(null); }
    public CalendarDeserializer(Class<? extends Calendar> cc) {
        super(Calendar.class);
        _calendarClass = cc;
    }

    @Override
```

```
    public Calendar deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        Date d = _parseDate(jp, ctxt);
        if (d == null) {
            return null;
        }
        if (_calendarClass == null) {
            return ctxt.constructCalendar(d);
        }
        try {
            Calendar c = _calendarClass.newInstance();
            c.setTimeInMillis(d.getTime());
            return c;
        } catch (Exception e) {
            throw ctxt.instantiationException(_calendarClass, e);
        }
    }
}

/**
 * Compared to plain old {@link java.util.Date}, SQL version is easier
 * to deal with: mostly because it is more limited.
 */
public static class SqlDateDeserializer
    extends StdScalarDeserializer<java.sql.Date>
{
    public SqlDateDeserializer() { super(java.sql.Date.class); }

    @Override
    public java.sql.Date deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        Date d = _parseDate(jp, ctxt);
        return (d == null) ? null : new java.sql.Date(d.getTime());
    }
}

/*
/**********************************************************
/* And other oddities
/**********************************************************
*/

public static class StackTraceElementDeserializer
    extends StdScalarDeserializer<StackTraceElement>
{
    public StackTraceElementDeserializer() { super(StackTraceElement.class); }
```

```
    @Override
    public StackTraceElement deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();
        // Must get an Object
        if (t == JsonToken.START_OBJECT) {
            String className = "", methodName = "", fileName = "";
            int lineNumber = -1;

            while ((t = jp.nextValue()) != JsonToken.END_OBJECT) {
                String propName = jp.getCurrentName();
                if ("className".equals(propName)) {
                    className = jp.getText();
                } else if ("fileName".equals(propName)) {
                    fileName = jp.getText();
                } else if ("lineNumber".equals(propName)) {
                    if (t.isNumeric()) {
                        lineNumber = jp.getIntValue();
                    } else {
                        throw JsonMappingException.from(jp, "Non-numeric token ("+t+") for property 'lineNumber'");
                    }
                } else if ("methodName".equals(propName)) {
                    methodName = jp.getText();
                } else if ("nativeMethod".equals(propName)) {
                    // no setter, not passed via constructor: ignore
                } else {
                    handleUnknownProperty(jp, ctxt, _valueClass, propName);
                }
            }
            return new StackTraceElement(className, methodName, fileName, lineNumber);
        }
        throw ctxt.mappingException(_valueClass);
    }
}

/**
 * We also want to directly support deserialization of
 * {@link TokenBuffer}.
 *<p>
 * Note that we use scalar deserializer base just because we claim
 * to be of scalar for type information inclusion purposes; actual
 * underlying content can be of any (Object, Array, scalar) type.
 *
 * @since 1.5
 */
@JacksonStdImpl
```

```
  public static class TokenBufferDeserializer
     extends StdScalarDeserializer<TokenBuffer>
  {
     public TokenBufferDeserializer() { super(TokenBuffer.class); }

     @Override
     public TokenBuffer deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
     {
        TokenBuffer tb = new TokenBuffer(jp.getCodec());
        // quite simple, given that TokenBuffer is a JsonGenerator:
        tb.copyCurrentStructure(jp);
        return tb;
     }
  }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.deser.BeanDeserializer;
import org.codehaus.jackson.map.introspect.AnnotatedClass;
import org.codehaus.jackson.map.type.*;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.type.JavaType;

/**
* Default {@link DeserializerProvider} implementation.
* Handles low-level caching (non-root) aspects of deserializer
* handling; all construction details are delegated to configured
*  {@link DeserializerFactory} instance that the provider owns.
*/
public class StdDeserializerProvider
  extends DeserializerProvider
{
  /*
  /**********************************************************
  /* Caching
  /**********************************************************
   */

  /**
```

```
 * Set of available key deserializers is currently limited
 * to standard types; and all known instances are storing
 * in this map.
 */
final static HashMap<JavaType, KeyDeserializer> _keyDeserializers = StdKeyDeserializers.constructAll();

/**
 * We will also cache some dynamically constructed deserializers;
 * specifically, ones that are expensive to construct.
 * This currently means bean and Enum deserializers; array, List and Map
 * deserializers will not be cached.
 *<p>
 * Given that we don't expect much concurrency for additions
 * (should very quickly converge to zero after startup), let's
 * explicitly define a low concurrency setting.
 */
final protected ConcurrentHashMap<JavaType, JsonDeserializer<Object>> _cachedDeserializers
   = new ConcurrentHashMap<JavaType, JsonDeserializer<Object>>(64, 0.75f, 2);

/**
 * During deserializer construction process we may need to keep track of partially
 * completed deserializers, to resolve cyclic dependencies. This is the
 * map used for storing deserializers before they are fully complete.
 */
final protected HashMap<JavaType, JsonDeserializer<Object>> _incompleteDeserializers
   = new HashMap<JavaType, JsonDeserializer<Object>>(8);

/*
/**********************************************************
/* Configuration
/**********************************************************
 */

/**
 * Factory responsible for constructing actual deserializers, if not
 * one of pre-configured types.
 */
protected DeserializerFactory _factory;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

/**
 * Default constructor. Equivalent to calling
 *<pre>
```

```
     *   new StdDeserializerProvider(BeanDeserializerFactory.instance);
     *</pre>
     */
    public StdDeserializerProvider() { this(BeanDeserializerFactory.instance); }

    public StdDeserializerProvider(DeserializerFactory f) {
        _factory = f;
    }

    @Override
    public DeserializerProvider withAdditionalDeserializers(Deserializers d) {
        _factory = _factory.withAdditionalDeserializers(d);
        return this;
    }

    @Override
    public DeserializerProvider withDeserializerModifier(BeanDeserializerModifier modifier) {
        _factory = _factory.withDeserializerModifier(modifier);
        return this;
    }

    /*
    /**********************************************************
    /* Abstract methods impls
    /**********************************************************
     */

    @SuppressWarnings("unchecked")
    @Override
    public JsonDeserializer<Object> findValueDeserializer(DeserializationConfig config,
            JavaType propertyType, BeanProperty property)
        throws JsonMappingException
    {
        JsonDeserializer<Object> deser = _findCachedDeserializer(propertyType);
        if (deser != null) {
            // [JACKSON-385]: need to support contextualization:
            if (deser instanceof ContextualDeserializer<?>) {
                JsonDeserializer<?> d = ((ContextualDeserializer<?>) deser).createContextual(config, property);
                deser = (JsonDeserializer<Object>) d;
            }
            return deser;
        }
        // If not, need to request factory to construct (or recycle)
        deser = _createAndCacheValueDeserializer(config, propertyType, property);
        if (deser == null) {
            /* Should we let caller handle it? Let's have a helper method
             * decide it; can throw an exception, or return a valid
             * deserializer
```

```
         */
        deser = _handleUnknownValueDeserializer(propertyType);
    }
    // [JACKSON-385]: need to support contextualization:
    if (deser instanceof ContextualDeserializer<?>) {
        JsonDeserializer<?> d = ((ContextualDeserializer<?>) deser).createContextual(config, property);
        deser = (JsonDeserializer<Object>) d;
    }
    return deser;
}

@Override
public JsonDeserializer<Object> findTypedValueDeserializer(DeserializationConfig config,
        JavaType type, BeanProperty property)
    throws JsonMappingException
{
    JsonDeserializer<Object> deser = findValueDeserializer(config, type, property);
    TypeDeserializer typeDeser = _factory.findTypeDeserializer(config, type, property);
    if (typeDeser != null) {
        return new WrappedDeserializer(typeDeser, deser);
    }
    return deser;
}


@Override
public KeyDeserializer findKeyDeserializer(DeserializationConfig config,
        JavaType type, BeanProperty property)
    throws JsonMappingException
{
    // No serializer needed if it's plain old String, or Object/untyped
    Class<?> raw = type.getRawClass();
    if (raw == String.class || raw == Object.class) {
        return null;
    }
    // Most other keys are of limited number of static types
    KeyDeserializer kdes = _keyDeserializers.get(type);
    if (kdes != null) {
        return kdes;
    }
    // And then other one-offs; first, Enum:
    if (type.isEnumType()) {
        return StdKeyDeserializers.constructEnumKeyDeserializer(config, type);
    }
    // One more thing: can we find ctor(String) or valueOf(String)?
    kdes = StdKeyDeserializers.findStringBasedKeyDeserializer(config, type);
    if (kdes != null) {
        return kdes;
```

```
    }

    // otherwise, will probably fail:
    return _handleUnknownKeyDeserializer(type);
}

/**
 * Method that can be called to find out whether a deserializer can
 * be found for given type
 */
@Override
public boolean hasValueDeserializerFor(DeserializationConfig config, JavaType type)
{
    /* Note: mostly copied from findValueDeserializer, except for
     * handling of unknown types
     */
    JsonDeserializer<Object> deser = _findCachedDeserializer(type);
    if (deser == null) {
        try {
            deser = _createAndCacheValueDeserializer(config, type, null);
        } catch (Exception e) {
            return false;
        }
    }
    return (deser != null);
}

@Override
public int cachedDeserializersCount() {
    return _cachedDeserializers.size();
}

/**
 * Method that will drop all dynamically constructed deserializers (ones that
 * are counted as result value for {@link #cachedDeserializersCount}).
 * This can be used to remove memory usage (in case some deserializers are
 * only used once or so), or to force re-construction of deserializers after
 * configuration changes for mapper than owns the provider.
 *
 * @since 1.4
 */
@Override
public void flushCachedDeserializers() {
    _cachedDeserializers.clear();
}

/*
/**********************************************************
```

```
/* Overridable helper methods
/**********************************************************
 */

protected JsonDeserializer<Object> _findCachedDeserializer(JavaType type)
{
   return _cachedDeserializers.get(type);
}

/**
 * Method that will try to create a deserializer for given type,
 * and resolve and cache it if necessary
 *
 * @param config Configuration
 * @param type Type of property to deserializer
 * @param property Property (field, setter, ctor arg) to use deserializer for
 */
protected JsonDeserializer<Object>_createAndCacheValueDeserializer(DeserializationConfig config,
      JavaType type, BeanProperty property)
   throws JsonMappingException
{
   /* Only one thread to construct deserializers at any given point in time;
    * limitations necessary to ensure that only completely initialized ones
    * are visible and used.
    */
   synchronized (_incompleteDeserializers) {
      // Ok, then: could it be that due to a race condition, deserializer can now be found?
      JsonDeserializer<Object> deser = _findCachedDeserializer(type);
      if (deser != null) {
         return deser;
      }
      int count = _incompleteDeserializers.size();
      // Or perhaps being resolved right now?
      if (count > 0) {
         deser = _incompleteDeserializers.get(type);
         if (deser != null) {
            return deser;
         }
      }
      // Nope: need to create and possibly cache
      try {
         return _createAndCache2(config, type, property);
      } finally {
         // also: any deserializers that have been created are complete by now
         if (count == 0 && _incompleteDeserializers.size() > 0) {
            _incompleteDeserializers.clear();
         }
      }
```

```
    }
}

/**
 * Method that handles actual construction (via factory) and caching (both
 * intermediate and eventual)
 */
protected JsonDeserializer<Object> _createAndCache2(DeserializationConfig config, JavaType type,
        BeanProperty property)
    throws JsonMappingException
{
    JsonDeserializer<Object> deser;
    try {
        deser = _createDeserializer(config, type, property);
    } catch (IllegalArgumentException iae) {
        /* We better only expose checked exceptions, since those
         * are what caller is expected to handle
         */
        throw new JsonMappingException(iae.getMessage(), null, iae);
    }
    if (deser == null) {
        return null;
    }
    /* cache resulting deserializer? always true for "plain" BeanDeserializer
     * (but can be re-defined for sub-classes by using @JsonCachable!)
     */
    // 08-Jun-2010, tatu: Related to [JACKSON-296], need to avoid caching MapSerializers... so:
    boolean isResolvable = (deser instanceof ResolvableDeserializer);
    boolean addToCache = (deser.getClass() == BeanDeserializer.class);
    if (!addToCache) {
        AnnotationIntrospector aintr = config.getAnnotationIntrospector();
        // note: pass 'null' to prevent mix-ins from being used
        AnnotatedClass ac = AnnotatedClass.construct(deser.getClass(), aintr, null);
        Boolean cacheAnn = aintr.findCachability(ac);
        if (cacheAnn != null) {
            addToCache = cacheAnn.booleanValue();
        }
    }
    /* we will temporarily hold on to all created deserializers (to
     * handle cyclic references, and possibly reuse non-cached
     * deserializers (list, map))
     */
    /* 07-Jun-2010, tatu: Danger: [JACKSON-296] was caused by accidental
     *   resolution of a reference -- couple of ways to prevent this;
     *   either not add Lists or Maps, or clear references eagerly.
     *   Let's actually do both; since both seem reasonable.
     */
    /* Need to resolve? Mostly done for bean deserializers; required for
```

```
     * resolving cyclic references.
     */
    if (isResolvable) {
        _incompleteDeserializers.put(type, deser);
        _resolveDeserializer(config, (ResolvableDeserializer)deser);
        _incompleteDeserializers.remove(type);
    }
    if (addToCache) {
        _cachedDeserializers.put(type, deser);
    }
    return deser;
}

/* Refactored so we can isolate the casts that require suppression
 * of type-safety warnings.
 */
@SuppressWarnings("unchecked")
protected JsonDeserializer<Object> _createDeserializer(DeserializationConfig config,
        JavaType type, BeanProperty property)
    throws JsonMappingException
{
    if (type.isEnumType()) {
        return (JsonDeserializer<Object>) _factory.createEnumDeserializer(config, this, type, property);
    }
    if (type.isContainerType()) {
        if (type instanceof ArrayType) {
            return (JsonDeserializer<Object>)_factory.createArrayDeserializer(config, this,
                    (ArrayType) type, property);
        }
        if (type instanceof MapType) {
            return (JsonDeserializer<Object>)_factory.createMapDeserializer(config, this,
                    (MapType) type, property);
        }
        if (type instanceof CollectionType) {
            return (JsonDeserializer<Object>)_factory.createCollectionDeserializer(config, this,
                    (CollectionType) type, property);
        }
    }

    // 02-Mar-2009, tatu: Let's consider JsonNode to be a type of its own
    if (JsonNode.class.isAssignableFrom(type.getRawClass())) {
        return (JsonDeserializer<Object>)_factory.createTreeDeserializer(config, this, type, property);
    }
    return (JsonDeserializer<Object>)_factory.createBeanDeserializer(config, this, type, property);
}

protected void _resolveDeserializer(DeserializationConfig config, ResolvableDeserializer ser)
    throws JsonMappingException
```

```
{
   ser.resolve(config, this);
}

/*
/**********************************************************
/* Overridable error reporting methods
/**********************************************************
 */

protected JsonDeserializer<Object> _handleUnknownValueDeserializer(JavaType type)
   throws JsonMappingException
{
   /* Let's try to figure out the reason, to give better error
    * messages
    */
   Class<?> rawClass = type.getRawClass();
   if (!ClassUtil.isConcrete(rawClass)) {
      throw new JsonMappingException("Can not find a Value deserializer for abstract type "+type);
   }
   throw new JsonMappingException("Can not find a Value deserializer for type "+type);
}

protected KeyDeserializer _handleUnknownKeyDeserializer(JavaType type)
   throws JsonMappingException
{
   throw new JsonMappingException("Can not find a (Map) Key deserializer for type "+type);
}

/*
/**********************************************************
/*  Helper classes
/**********************************************************
 */

/**
 * Simple deserializer that will call configured type deserializer, passing
 * in configured data deserializer, and exposing it all as a simple
 * deserializer.
 */
protected final static class WrappedDeserializer
   extends JsonDeserializer<Object>
{
   final TypeDeserializer _typeDeserializer;
   final JsonDeserializer<Object> _deserializer;

   public WrappedDeserializer(TypeDeserializer typeDeser, JsonDeserializer<Object> deser)
   {
```

```
        super();
        _typeDeserializer = typeDeser;
        _deserializer = deser;
    }

    @Override
    public Object deserialize(JsonParser jp, DeserializationContext ctxt)
            throws IOException, JsonProcessingException
    {
        return _deserializer.deserializeWithType(jp, ctxt, _typeDeserializer);
    }

    @Override
    public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
        TypeDeserializer typeDeserializer)
            throws IOException, JsonProcessingException
    {
        // should never happen? (if it can, could call on that object)
        throw new IllegalStateException("Type-wrapped deserializer's deserializeWithType should never get
called");
    }
  }

}
package org.codehaus.jackson.map.deser;

import java.lang.reflect.*;
import java.util.*;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.annotate.JsonAutoDetect.Visibility;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.introspect.*;
import org.codehaus.jackson.map.type.*;
import org.codehaus.jackson.map.util.ArrayBuilders;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.type.JavaType;

/**
* Concrete deserializer factory class that adds full Bean deserializer
* construction logic using class introspection.
*<p>
* Since there is no caching, this factory is stateless and a globally
* shared singleton instance ({@link #instance}) can be  used by
* {@link DeserializerProvider}s).
*/
public class BeanDeserializerFactory
  extends BasicDeserializerFactory
```

```
{
  /**
   * Signature of <b>Throwable.initCause</b> method.
   */
  private final static Class<?>[] INIT_CAUSE_PARAMS = new Class<?>[] { Throwable.class };

  /*
  /**********************************************************
  /* Config class implementation
  /**********************************************************
   */

  /**
   * Standard configuration settings container class implementation.
   *
   * @since 1.7
   */
  public static class ConfigImpl extends Config
  {
     protected final static BeanDeserializerModifier[] NO_MODIFIERS = new BeanDeserializerModifier[0];

     /**
      * List of providers for additional deserializers, checked before considering default
      * basic or bean deserialializers.
      *
      * @since 1.7
      */
     protected final Deserializers[] _additionalDeserializers;

     /**
      * List of modifiers that can change the way {@link BeanDeserializer} instances
      * are configured and constructed.
      */
     protected final BeanDeserializerModifier[] _modifiers;

     /**
      * Constructor for creating basic configuration with no additional
      * handlers.
      */
     public ConfigImpl() {
        this(null, null);
     }

     /**
      * Copy-constructor that will create an instance that contains defined
      * set of additional deserializer providers.
      */
     protected ConfigImpl(Deserializers[] allAdditionalDeserializers,
```

```
            BeanDeserializerModifier[] modifiers)
    {
        _additionalDeserializers = (allAdditionalDeserializers == null) ?
                NO_DESERIALIZERS : allAdditionalDeserializers;
        _modifiers = (modifiers == null) ? NO_MODIFIERS : modifiers;
    }

    @Override
    public Config withAdditionalDeserializers(Deserializers additional)
    {
        if (additional == null) {
            throw new IllegalArgumentException("Can not pass null Deserializers");
        }
        Deserializers[] all = ArrayBuilders.insertInList(_additionalDeserializers, additional);
        return new ConfigImpl(all, _modifiers);
    }

    @Override
    public Config withDeserializerModifier(BeanDeserializerModifier modifier)
    {
        if (modifier == null) {
            throw new IllegalArgumentException("Can not pass null modifier");
        }
        BeanDeserializerModifier[] all = ArrayBuilders.insertInList(_modifiers, modifier);
        return new ConfigImpl(_additionalDeserializers, all);
    }

    @Override
    public boolean hasDeserializers() { return _additionalDeserializers.length > 0; }

    @Override
    public boolean hasDeserializerModifiers() { return _modifiers.length > 0; }

    @Override
    public Iterable<Deserializers> deserializers() {
        return ArrayBuilders.arrayAsIterable(_additionalDeserializers);
    }

    @Override
    public Iterable<BeanDeserializerModifier> deserializerModifiers() {
        return ArrayBuilders.arrayAsIterable(_modifiers);
    }
}

/*
/**********************************************************
/* Life-cycle
/**********************************************************
```

```
 */

/**
 * Globally shareable thread-safe instance which has no additional custom deserializers
 * registered
 */
public final static BeanDeserializerFactory instance = new BeanDeserializerFactory(null);

/**
 * Configuration settings for this factory; immutable instance (just like this
 * factory), new version created via copy-constructor (fluent-style)
 *
 * @since 1.7
 */
protected final Config _factoryConfig;

@Deprecated
public BeanDeserializerFactory() {
    this(null);
}

/**
 * @since 1.7
 */
public BeanDeserializerFactory(DeserializerFactory.Config config) {
    if (config == null) {
        config = new ConfigImpl();
    }
    _factoryConfig = config;
}

@Override
public final Config getConfig() {
    return _factoryConfig;
}

/**
 * Method used by module registration functionality, to construct a new bean
 * deserializer factory
 * with different configuration settings.
 *
 * @since 1.7
 */
@Override
public DeserializerFactory withConfig(DeserializerFactory.Config config)
{
    if (_factoryConfig == config) {
        return this;
```

```
    }

    /* 22-Nov-2010, tatu: Handling of subtypes is tricky if we do immutable-with-copy-ctor;
     *    and we pretty much have to here either choose between losing subtype instance
     *    when registering additional deserializers, or losing deserializers.
     *    Instead, let's actually just throw an error if this method is called when subtype
     *    has not properly overridden this method; this to indicate problem as soon as possible.
     */
    if (getClass() != BeanDeserializerFactory.class) {
        throw new IllegalStateException("Subtype of BeanDeserializerFactory ("+getClass().getName()
                +") has not properly overridden method 'withAdditionalDeserializers': can not instantiate subtype with "
                +"additional deserializer definitions");
    }
    return new BeanDeserializerFactory(config);
}

/*
/**********************************************************
/* Overrides for super-class methods used for finding
/* custom deserializers
/**********************************************************
 */

@Override
protected JsonDeserializer<?> _findCustomArrayDeserializer(ArrayType type, DeserializationConfig config,
        DeserializerProvider provider,
        BeanProperty property,
        TypeDeserializer elementTypeDeserializer, JsonDeserializer<?> elementDeserializer)
    throws JsonMappingException
{
    for (Deserializers d  : _factoryConfig.deserializers()) {
        JsonDeserializer<?> deser = d.findArrayDeserializer(type, config, provider, property,
                elementTypeDeserializer, elementDeserializer);
        if (deser != null) {
            return deser;
        }
    }
    return null;
}

@Override
protected JsonDeserializer<?> _findCustomCollectionDeserializer(CollectionType type, DeserializationConfig
config,
        DeserializerProvider provider, BasicBeanDescription beanDesc,
        BeanProperty property,
        TypeDeserializer elementTypeDeserializer, JsonDeserializer<?> elementDeserializer)
    throws JsonMappingException
{
```

```
    for (Deserializers d  : _factoryConfig.deserializers()) {
        JsonDeserializer<?> deser = d.findCollectionDeserializer(type, config, provider, beanDesc, property,
                elementTypeDeserializer, elementDeserializer);
        if (deser != null) {
            return deser;
        }
    }
    return null;
}

@Override
protected JsonDeserializer<?> _findCustomEnumDeserializer(Class<?> type, DeserializationConfig config,
        BasicBeanDescription beanDesc, BeanProperty property)
    throws JsonMappingException
{
    for (Deserializers d  : _factoryConfig.deserializers()) {
        JsonDeserializer<?> deser = d.findEnumDeserializer(type, config, beanDesc, property);
        if (deser != null) {
            return deser;
        }
    }
    return null;
}

@Override
protected JsonDeserializer<?> _findCustomMapDeserializer(MapType type, DeserializationConfig config,
        DeserializerProvider provider, BasicBeanDescription beanDesc, BeanProperty property,
        KeyDeserializer keyDeserializer,
        TypeDeserializer elementTypeDeserializer, JsonDeserializer<?> elementDeserializer)
    throws JsonMappingException
{
    for (Deserializers d  : _factoryConfig.deserializers()) {
        JsonDeserializer<?> deser = d.findMapDeserializer(type, config, provider, beanDesc, property,
                keyDeserializer, elementTypeDeserializer, elementDeserializer);
        if (deser != null) {
            return deser;
        }
    }
    return null;
}

@Override
protected JsonDeserializer<?> _findCustomTreeNodeDeserializer(Class<? extends JsonNode> type,
        DeserializationConfig config, BeanProperty property)
    throws JsonMappingException
{
    for (Deserializers d  : _factoryConfig.deserializers()) {
        JsonDeserializer<?> deser = d.findTreeNodeDeserializer(type, config, property);
```

```
            if (deser != null) {
                return deser;
            }
        }
        return null;
    }

    // Note: NOT overriding, superclass has no matching method
    @SuppressWarnings("unchecked")
    protected JsonDeserializer<Object> _findCustomBeanDeserializer(JavaType type, DeserializationConfig config,
            DeserializerProvider provider, BasicBeanDescription beanDesc, BeanProperty property)
        throws JsonMappingException
    {
        for (Deserializers d  : _factoryConfig.deserializers()) {
            JsonDeserializer<?> deser = d.findBeanDeserializer(type, config, provider, beanDesc, property);
            if (deser != null) {
                return (JsonDeserializer<Object>) deser;
            }
        }
        return null;
    }

    /*
    /**********************************************************
    /* DeserializerFactory API implementation
    /**********************************************************
     */

    /**
     * Method that {@link DeserializerProvider}s call to create a new
     * deserializer for types other than Collections, Maps, arrays and
     * enums.
     */
    @Override
    public JsonDeserializer<Object> createBeanDeserializer(DeserializationConfig config, DeserializerProvider p,
            JavaType type, BeanProperty property)
        throws JsonMappingException
    {
        // First things first: maybe explicit definition via annotations?
        BasicBeanDescription beanDesc = config.introspect(type);
        JsonDeserializer<Object> ad = findDeserializerFromAnnotation(config, beanDesc.getClassInfo(), property);
        if (ad != null) {
            return ad;
        }
        // Or value annotation that indicates more specific type to use:
        JavaType newType =  modifyTypeByAnnotation(config, beanDesc.getClassInfo(), type, null);
        if (newType.getRawClass() != type.getRawClass()) {
            type = newType;
```

```
    beanDesc = config.introspect(type);
}
// We may also have custom overrides:
JsonDeserializer<Object> custom = _findCustomBeanDeserializer(type, config, p, beanDesc, property);
if (custom != null) {
    return custom;
}

/* Let's call super class first: it knows simple types for
 * which we have default deserializers
 */
JsonDeserializer<Object> deser = super.createBeanDeserializer(config, p, type, property);
if (deser != null) {
    return deser;
}
// Otherwise: could the class be a Bean class?
if (!isPotentialBeanType(type.getRawClass())) {
    return null;
}

/* One more thing to check: do we have an exception type
 * (Throwable or its sub-classes)? If so, need slightly
 * different handling.
 */
if (type.isThrowable()) {
    return buildThrowableDeserializer(config, type, beanDesc, property);
}

/* Abstract types can not be handled using regular bean deserializer, but need
 * either type deserializer (which caller usually deals with), or resolve
 * to a concrete type.
 */
if (type.isAbstract()) {
    // [JACKSON-41] (v1.6): Let's make it possible to materialize abstract types.
    /* One check however: if type information is indicated, we usually do not
     * want materialization...
     */
    AbstractTypeResolver res = config.getAbstractTypeResolver();
    if (res != null) {
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
        if (intr.findTypeResolver(beanDesc.getClassInfo(), type) == null) {
            JavaType concrete = res.resolveAbstractType(config, type);
            if (concrete != null) {
                /* important: introspect actual implementation (abstract class or
                 * interface doesn't have constructors, for one)
                 */
                beanDesc = config.introspect(concrete);
                return buildBeanDeserializer(config, concrete, beanDesc, property);
```

```
            }
          }
        }
        // otherwise we assume there must be extra type information coming
        return new AbstractDeserializer(type);
    }

    /* Otherwise we'll just use generic bean introspection
     * to build deserializer
     */
    return buildBeanDeserializer(config, type, beanDesc, property);
}

/*
/**********************************************************
/* Public construction method beyond DeserializerFactory API:
/* can be called from outside as well as overridden by
/* sub-classes
/**********************************************************
 */

/**
 * Method that is to actually build a bean deserializer instance.
 * All basic sanity checks have been done to know that what we have
 * may be a valid bean type, and that there are no default simple
 * deserializers.
 */
@SuppressWarnings("unchecked")
public JsonDeserializer<Object> buildBeanDeserializer(DeserializationConfig config,
        JavaType type, BasicBeanDescription beanDesc, BeanProperty property)
    throws JsonMappingException
{
    BeanDeserializerBuilder builder = constructBeanDeserializerBuilder(beanDesc);
    builder.setCreators(findDeserializerCreators(config, beanDesc));
     // And then setters for deserializing from JSON Object
    addBeanProps(config, beanDesc, builder);
    // managed/back reference fields/setters need special handling... first part
    addReferenceProperties(config, beanDesc, builder);

    // [JACKSON-440]: update builder now that all information is in?
    if (_factoryConfig.hasDeserializerModifiers()) {
        for (BeanDeserializerModifier mod : _factoryConfig.deserializerModifiers()) {
            builder = mod.updateBuilder(config, beanDesc, builder);
        }
    }
    JsonDeserializer<?> deserializer = builder.build(property);

    // [JACKSON-440]: may have modifier(s) that wants to modify or replace serializer we just built:
```

```
    if (_factoryConfig.hasDeserializerModifiers()) {
        for (BeanDeserializerModifier mod : _factoryConfig.deserializerModifiers()) {
            deserializer = mod.modifyDeserializer(config, beanDesc, deserializer);
        }
    }
    return (JsonDeserializer<Object>) deserializer;

}

@SuppressWarnings("unchecked")
public JsonDeserializer<Object> buildThrowableDeserializer(DeserializationConfig config,
        JavaType type, BasicBeanDescription beanDesc, BeanProperty property)
    throws JsonMappingException
{
    // first: construct like a regular bean deserializer...
    BeanDeserializerBuilder builder = constructBeanDeserializerBuilder(beanDesc);
    builder.setCreators(findDeserializerCreators(config, beanDesc));

    addBeanProps(config, beanDesc, builder);
    // (and assume there won't be any back references)

    // But then let's decorate things a bit
    /* To resolve [JACKSON-95], need to add "initCause" as setter
     * for exceptions (sub-classes of Throwable).
     */
    AnnotatedMethod am = beanDesc.findMethod("initCause", INIT_CAUSE_PARAMS);
    if (am != null) { // should never be null
        SettableBeanProperty prop = constructSettableProperty(config, beanDesc, "cause", am);
        if (prop != null) {
            builder.addProperty(prop);
        }
    }

    // And also need to ignore "localizedMessage"
    builder.addIgnorable("localizedMessage");
    /* As well as "message": it will be passed via constructor,
     * as there's no 'setMessage()' method
    */
    builder.addIgnorable("message");

    // [JACKSON-440]: update builder now that all information is in?
    if (_factoryConfig.hasDeserializerModifiers()) {
        for (BeanDeserializerModifier mod : _factoryConfig.deserializerModifiers()) {
            builder = mod.updateBuilder(config, beanDesc, builder);
        }
    }
    JsonDeserializer<?> deserializer = builder.build(property);
```

```
        /* At this point it ought to be a BeanDeserializer; if not, must assume
         * it's some other thing that can handle deserialization ok...
         */
        if (deserializer instanceof BeanDeserializer) {
            deserializer = new ThrowableDeserializer((BeanDeserializer) deserializer);
        }

        // [JACKSON-440]: may have modifier(s) that wants to modify or replace serializer we just built:
        if (_factoryConfig.hasDeserializerModifiers()) {
            for (BeanDeserializerModifier mod : _factoryConfig.deserializerModifiers()) {
                deserializer = mod.modifyDeserializer(config, beanDesc, deserializer);
            }
        }
        return (JsonDeserializer<Object>) deserializer;
    }

    /*
    /**********************************************************
    /* Helper methods for Bean deserializer construction,
    /* overridable by sub-classes
    /**********************************************************
     */

    /**
     * Overridable method that constructs a {@link BeanDeserializerBuilder}
     * which is used to accumulate information needed to create deserializer
     * instance.
     *
     * @since 1.7
     */
    protected BeanDeserializerBuilder constructBeanDeserializerBuilder(BasicBeanDescription beanDesc) {
        return new BeanDeserializerBuilder(beanDesc);
    }

    /**
     * Method that is to find all creators (constructors, factory methods)
     * for the bean type to deserialize.
     *
     * @since 1.7
     */
    protected CreatorContainer findDeserializerCreators(DeserializationConfig config,
            BasicBeanDescription beanDesc)
        throws JsonMappingException
    {
        boolean fixAccess =
    config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS);
        CreatorContainer creators =  new CreatorContainer(beanDesc.getBeanClass(), fixAccess);
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
```

```
    // First, let's figure out constructor/factory-based instantation
    // 23-Jan-2010, tatus: but only for concrete types
    if (beanDesc.getType().isConcrete()) {
        Constructor<?> defaultCtor = beanDesc.findDefaultConstructor();
        if (defaultCtor != null) {
            if (fixAccess) {
                ClassUtil.checkAndFixAccess(defaultCtor);
            }

            creators.setDefaultConstructor(defaultCtor);
        }
    }

    // need to construct suitable visibility checker:
    VisibilityChecker<?> vchecker = config.getDefaultVisibilityChecker();
    if (!config.isEnabled(DeserializationConfig.Feature.AUTO_DETECT_CREATORS)) {
        vchecker = vchecker.withCreatorVisibility(Visibility.NONE);
    }
    vchecker = config.getAnnotationIntrospector().findAutoDetectVisibility(beanDesc.getClassInfo(), vchecker);

    _addDeserializerConstructors(config, beanDesc, vchecker, intr, creators);
    _addDeserializerFactoryMethods(config, beanDesc, vchecker, intr, creators);
    return creators;
}

protected void _addDeserializerConstructors
    (DeserializationConfig config, BasicBeanDescription beanDesc, VisibilityChecker<?> vchecker,
     AnnotationIntrospector intr, CreatorContainer creators)
    throws JsonMappingException
{
    for (AnnotatedConstructor ctor : beanDesc.getConstructors()) {
        int argCount = ctor.getParameterCount();
        if (argCount < 1) {
            continue;
        }
        boolean isCreator = intr.hasCreatorAnnotation(ctor);
        // some single-arg constructors (String, number) are auto-detected
        if (argCount == 1) {
            /* but note: if we do have parameter name, it'll be
             * "property constructor", and needs to be skipped for now
             */
            String name = intr.findPropertyNameForParam(ctor.getParameter(0));
            if (name == null || name.length() == 0) { // not property based
                Class<?> type = ctor.getParameterClass(0);
                if (type == String.class) {
                    if (isCreator || vchecker.isCreatorVisible(ctor)) {
                        creators.addStringConstructor(ctor);
```

```
                }
                continue;
            }
            if (type == int.class || type == Integer.class) {
                if (isCreator || vchecker.isCreatorVisible(ctor)) {
                    creators.addIntConstructor(ctor);
                }
                continue;
            }
            if (type == long.class || type == Long.class) {
                if (isCreator || vchecker.isCreatorVisible(ctor)) {
                    creators.addLongConstructor(ctor);
                }
                continue;
            }
            // Delegating constructor ok iff it has @JsonCreator (etc)
            if (intr.hasCreatorAnnotation(ctor)) {
                creators.addDelegatingConstructor(ctor);
            }
            // otherwise just ignored
            continue;
        }
        // fall through if there's name
    } else {
        // more than 2 args, must be @JsonCreator
        if (!intr.hasCreatorAnnotation(ctor)) {
            continue;
        }
    }

    // 1 or more args; all params must have name annotations
    SettableBeanProperty[] properties = new SettableBeanProperty[argCount];
    for (int i = 0; i < argCount; ++i) {
        AnnotatedParameter param = ctor.getParameter(i);
        String name = (param == null) ? null : intr.findPropertyNameForParam(param);
        // At this point, name annotation is NOT optional
        if (name == null || name.length() == 0) {
            throw new IllegalArgumentException("Argument #"+i+" of constructor "+ctor+" has no property name
annotation; must have name when multiple-paramater constructor annotated as Creator");
        }
        properties[i] = constructCreatorProperty(config, beanDesc, name, i, param);
    }
    creators.addPropertyConstructor(ctor, properties);
  }
}

protected void _addDeserializerFactoryMethods
  (DeserializationConfig config, BasicBeanDescription beanDesc, VisibilityChecker<?> vchecker,
```

```
    AnnotationIntrospector intr, CreatorContainer creators)
    throws JsonMappingException
{

    for (AnnotatedMethod factory : beanDesc.getFactoryMethods()) {
        int argCount = factory.getParameterCount();
        if (argCount < 1) {
            continue;
        }
        boolean isCreator = intr.hasCreatorAnnotation(factory);
        // some single-arg factory methods (String, number) are auto-detected
        if (argCount == 1) {
            /* but as above: if we do have parameter name, it'll be
             * "property constructor", and needs to be skipped for now
             */
            String name = intr.findPropertyNameForParam(factory.getParameter(0));
            if (name == null || name.length() == 0) { // not property based
                Class<?> type = factory.getParameterClass(0);
                if (type == String.class) {
                    if (isCreator || vchecker.isCreatorVisible(factory)) {
                        creators.addStringFactory(factory);
                    }
                    continue;
                }
                if (type == int.class || type == Integer.class) {
                    if (isCreator || vchecker.isCreatorVisible(factory)) {
                        creators.addIntFactory(factory);
                    }
                    continue;
                }
                if (type == long.class || type == Long.class) {
                    if (isCreator || vchecker.isCreatorVisible(factory)) {
                        creators.addLongFactory(factory);
                    }
                    continue;
                }
                if (intr.hasCreatorAnnotation(factory)) {
                    creators.addDelegatingFactory(factory);
                }
                // otherwise just ignored
                continue;
            }
            // fall through if there's name
        } else {
            // more than 2 args, must be @JsonCreator
            if (!intr.hasCreatorAnnotation(factory)) {
                continue;
            }
```

```
            }
            // 1 or more args; all params must have name annotations
            SettableBeanProperty[] properties = new SettableBeanProperty[argCount];
            for (int i = 0; i < argCount; ++i) {
                AnnotatedParameter param = factory.getParameter(i);
                String name = intr.findPropertyNameForParam(param);
                // At this point, name annotation is NOT optional
                if (name == null || name.length() == 0) {
                    throw new IllegalArgumentException("Argument #"+i+" of factory method "+factory+" has no property
name annotation; must have when multiple-paramater static method annotated as Creator");
                }
                properties[i] = constructCreatorProperty(config, beanDesc, name, i, param);
            }
            creators.addPropertyFactory(factory, properties);
        }
    }

    /**
     * Method called to figure out settable properties for the
     * bean deserializer to use.
     *<p>
     * Note: designed to be overridable, and effort is made to keep interface
     * similar between versions.
     */
    protected void addBeanProps(DeserializationConfig config,
            BasicBeanDescription beanDesc, BeanDeserializerBuilder builder)
        throws JsonMappingException
    {
        // Ok: let's aggregate visibility settings: first, baseline:
        VisibilityChecker<?> vchecker = config.getDefaultVisibilityChecker();
        // then global overrides (disabling)
        if (!config.isEnabled(DeserializationConfig.Feature.AUTO_DETECT_SETTERS)) {
            vchecker = vchecker.withSetterVisibility(Visibility.NONE);
        }
        if (!config.isEnabled(DeserializationConfig.Feature.AUTO_DETECT_FIELDS)) {
            vchecker = vchecker.withFieldVisibility(Visibility.NONE);
        }
        // and finally per-class overrides:
        vchecker = config.getAnnotationIntrospector().findAutoDetectVisibility(beanDesc.getClassInfo(), vchecker);

        Map<String,AnnotatedMethod> setters = beanDesc.findSetters(vchecker);
        // Also, do we have a fallback "any" setter?
        AnnotatedMethod anySetter = beanDesc.findAnySetter();

        // Things specified as "ok to ignore"? [JACKSON-77]
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
        boolean ignoreAny = false;
        {
```

```
   Boolean B = intr.findIgnoreUnknownProperties(beanDesc.getClassInfo());
   if (B != null) {
      ignoreAny = B.booleanValue();
      builder.setIgnoreUnknownProperties(ignoreAny);
   }
}
// Or explicit/implicit definitions?
HashSet<String> ignored = ArrayBuilders.arrayToSet(intr.findPropertiesToIgnore(beanDesc.getClassInfo()));

// But let's only add these if we'd otherwise fail with exception (save some memory here)
/* 03-Oct-2010, tatu: As per [JACKSON-383], we can't optimize here,
 *   since doing so may interfere with handling of @JsonAnySetter.
 */
//if (!ignoreAny && config.isEnabled(DeserializationConfig.Feature.FAIL_ON_UNKNOWN_PROPERTIES))
{
   for (String propName : ignored) {
      builder.addIgnorable(propName);
   }
   // Implicit ones via @JsonIgnore and equivalent?
   AnnotatedClass ac = beanDesc.getClassInfo();
   for (AnnotatedMethod am : ac.ignoredMemberMethods()) {
      String name = beanDesc.okNameForSetter(am);
      if (name != null) {
         builder.addIgnorable(name);
      }
   }
   for (AnnotatedField af : ac.ignoredFields()) {
      builder.addIgnorable(af.getName());
   }
}

HashMap<Class<?>,Boolean> ignoredTypes = new HashMap<Class<?>,Boolean>();

// These are all valid setters, but we do need to introspect bit more
for (Map.Entry<String,AnnotatedMethod> en : setters.entrySet()) {
   String name = en.getKey();
   if (!ignored.contains(name)) { // explicit ignoral using @JsonIgnoreProperties needs to block entries
      AnnotatedMethod setter = en.getValue();
      // [JACKSON-429] Some types are declared as ignorable as well
      Class<?> type = setter.getParameterClass(0);
      if (isIgnorableType(config, beanDesc, type, ignoredTypes)) {
         // important: make ignorable, to avoid errors if value is actually seen
         builder.addIgnorable(name);
         continue;
      }
      SettableBeanProperty prop = constructSettableProperty(config, beanDesc, name, setter);
      if (prop != null) {
         builder.addProperty(prop);
```

```
            }
          }
        }
        if (anySetter != null) {
          builder.setAnySetter(constructAnySetter(config, beanDesc, anySetter));
        }

        HashSet<String> addedProps = new HashSet<String>(setters.keySet());
        /* [JACKSON-98]: also include field-backed properties:
         *   (second arg passed to ignore anything for which there is a getter
         *   method)
         */
        LinkedHashMap<String,AnnotatedField> fieldsByProp = beanDesc.findDeserializableFields(vchecker,
addedProps);
        for (Map.Entry<String,AnnotatedField> en : fieldsByProp.entrySet()) {
          String name = en.getKey();
          if (!ignored.contains(name) && !builder.hasProperty(name)) {
            AnnotatedField field = en.getValue();
            // [JACKSON-429] Some types are declared as ignorable as well
            Class<?> type = field.getRawType();
            if (isIgnorableType(config, beanDesc, type, ignoredTypes)) {
              // important: make ignorable, to avoid errors if value is actually seen
              builder.addIgnorable(name);
              continue;
            }
            SettableBeanProperty prop = constructSettableProperty(config, beanDesc, name, field);
            if (prop != null) {
              builder.addProperty(prop);
              addedProps.add(name);
            }
          }
        }

        /* As per [JACKSON-88], may also need to consider getters
         * for Map/Collection properties
         */
        /* also, as per [JACKSON-328], should not override fields (or actual setters),
         * thus these are added AFTER adding fields
         */
        if (config.isEnabled(DeserializationConfig.Feature.USE_GETTERS_AS_SETTERS)) {
          /* Hmmh. We have to assume that 'use getters as setters' also
           * implies 'yes, do auto-detect these getters'? (if not, we'd
           * need to add AUTO_DETECT_GETTERS to deser config too, not
           * just ser config)
           */
          Map<String,AnnotatedMethod> getters = beanDesc.findGetters(vchecker, addedProps);

          for (Map.Entry<String,AnnotatedMethod> en : getters.entrySet()) {
```

```
            AnnotatedMethod getter = en.getValue();
            // should only consider Collections and Maps, for now?
            Class<?> rt = getter.getRawType();
            if (!Collection.class.isAssignableFrom(rt) && !Map.class.isAssignableFrom(rt)) {
               continue;
            }
            String name = en.getKey();
            if (!ignored.contains(name) && !builder.hasProperty(name)) {
               builder.addProperty(constructSetterlessProperty(config, beanDesc, name, getter));
               addedProps.add(name);
            }
         }
      }

   }

   /**
    * Method that will find if bean has any managed- or back-reference properties,
    * and if so add them to bean, to be linked during resolution phase.
    *
    * @since 1.6
    */
   protected void addReferenceProperties(DeserializationConfig config,
         BasicBeanDescription beanDesc, BeanDeserializerBuilder builder)
      throws JsonMappingException
   {
      // and then back references, not necessarily found as regular properties
      Map<String,AnnotatedMember> refs = beanDesc.findBackReferenceProperties();
      if (refs != null) {
         for (Map.Entry<String, AnnotatedMember> en : refs.entrySet()) {
            String name = en.getKey();
            AnnotatedMember m = en.getValue();
            if (m instanceof AnnotatedMethod) {
               builder.addBackReferenceProperty(name, constructSettableProperty(
                     config, beanDesc, m.getName(), (AnnotatedMethod) m));
            } else {
               builder.addBackReferenceProperty(name, constructSettableProperty(
                     config, beanDesc, m.getName(), (AnnotatedField) m));
            }
         }
      }
   }

   /**
    * Method called to construct fallback {@link SettableAnyProperty}
    * for handling unknown bean properties, given a method that
    * has been designated as such setter.
    */
```

```
  protected SettableAnyProperty constructAnySetter(DeserializationConfig config,
        BasicBeanDescription beanDesc, AnnotatedMethod setter)
    throws JsonMappingException
  {
    if (config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS)) {
      setter.fixAccess(); // to ensure we can call it
    }
    // we know it's a 2-arg method, second arg is the value
    JavaType type = TypeFactory.type(setter.getParameterType(1), beanDesc.bindingsForBeanType());
    BeanProperty.Std property = new BeanProperty.Std(setter.getName(), type, beanDesc.getClassAnnotations(),
setter);
    type = resolveType(config, beanDesc, type, setter, property);

    /* AnySetter can be annotated with @JsonClass (etc) just like a
     * regular setter... so let's see if those are used.
     * Returns null if no annotations, in which case binding will
     * be done at a later point.
     */
    JsonDeserializer<Object> deser = findDeserializerFromAnnotation(config, setter, property);
    if (deser != null) {
      SettableAnyProperty prop = new SettableAnyProperty(property, setter, type);
      prop.setValueDeserializer(deser);
      return prop;
    }
    /* Otherwise, method may specify more specific (sub-)class for
     * value (no need to check if explicit deser was specified):
     */
    type = modifyTypeByAnnotation(config, setter, type, property.getName());
    return new SettableAnyProperty(property, setter, type);
  }

  /**
   * Method that will construct a regular bean property setter using
   * the given setter method.
   *
   * @param setter Method to use to set property value; or null if none.
   *    Null only for "setterless" properties
   *
   * @return Property constructed, if any; or null to indicate that
   *   there should be no property based on given definitions.
   */
  protected SettableBeanProperty constructSettableProperty(DeserializationConfig config,
        BasicBeanDescription beanDesc, String name,
        AnnotatedMethod setter)
    throws JsonMappingException
  {
    // need to ensure method is callable (for non-public)
    if (config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS)) {
```

```
            setter.fixAccess();
        }

        // note: this works since we know there's exactly one arg for methods
        JavaType t0 = TypeFactory.type(setter.getParameterType(0), beanDesc.bindingsForBeanType());
        BeanProperty.Std property = new BeanProperty.Std(name, t0, beanDesc.getClassAnnotations(), setter);
        JavaType type = resolveType(config, beanDesc, t0, setter, property);
        // did type change?
        if (type != t0) {
            property = property.withType(type);
        }

        /* First: does the Method specify the deserializer to use?
         * If so, let's use it.
         */
        JsonDeserializer<Object> propDeser = findDeserializerFromAnnotation(config, setter, property);
        type = modifyTypeByAnnotation(config, setter, type, name);
        TypeDeserializer typeDeser = type.getTypeHandler();
        SettableBeanProperty prop = new SettableBeanProperty.MethodProperty(name, type, typeDeser,
                beanDesc.getClassAnnotations(), setter);
        if (propDeser != null) {
            prop.setValueDeserializer(propDeser);
        }
        // [JACKSON-235]: need to retain name of managed forward references:
        AnnotationIntrospector.ReferenceProperty ref = config.getAnnotationIntrospector().findReferenceType(setter);
        if (ref != null && ref.isManagedReference()) {
            prop.setManagedReferenceName(ref.getName());
        }
        return prop;
    }

    protected SettableBeanProperty constructSettableProperty(DeserializationConfig config,
            BasicBeanDescription beanDesc, String name, AnnotatedField field)
        throws JsonMappingException
    {
        // need to ensure method is callable (for non-public)
        if (config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS)) {
            field.fixAccess();
        }
        JavaType t0 = TypeFactory.type(field.getGenericType(), beanDesc.bindingsForBeanType());
        BeanProperty.Std property = new BeanProperty.Std(name, t0, beanDesc.getClassAnnotations(), field);
        JavaType type = resolveType(config, beanDesc, t0, field, property);
        // did type change?
        if (type != t0) {
            property = property.withType(type);
        }
        /* First: does the Method specify the deserializer to use?
         * If so, let's use it.
```

```
     */
    JsonDeserializer<Object> propDeser = findDeserializerFromAnnotation(config, field, property);
    type = modifyTypeByAnnotation(config, field, type, name);
    TypeDeserializer typeDeser = type.getTypeHandler();
    SettableBeanProperty prop = new SettableBeanProperty.FieldProperty(name, type, typeDeser,
            beanDesc.getClassAnnotations(), field);
    if (propDeser != null) {
        prop.setValueDeserializer(propDeser);
    }
    // [JACKSON-235]: need to retain name of managed forward references:
    AnnotationIntrospector.ReferenceProperty ref = config.getAnnotationIntrospector().findReferenceType(field);
    if (ref != null && ref.isManagedReference()) {
        prop.setManagedReferenceName(ref.getName());
    }
    return prop;
}

/**
 * Method that will construct a regular bean property setter using
 * the given setter method.
 *
 * @param getter Method to use to get property value to modify, null if
 *    none. Non-null for "setterless" properties.
 */
protected SettableBeanProperty constructSetterlessProperty(DeserializationConfig config,
        BasicBeanDescription beanDesc, String name, AnnotatedMethod getter)
    throws JsonMappingException
{
    // need to ensure it is callable now:
    if (config.isEnabled(DeserializationConfig.Feature.CAN_OVERRIDE_ACCESS_MODIFIERS)) {
        getter.fixAccess();
    }

    JavaType type = getter.getType(beanDesc.bindingsForBeanType());
    /* First: does the Method specify the deserializer to use?
     * If so, let's use it.
     */
    BeanProperty.Std property = new BeanProperty.Std(name, type, beanDesc.getClassAnnotations(), getter);
    // @TODO: create BeanProperty to pass?
    JsonDeserializer<Object> propDeser = findDeserializerFromAnnotation(config, getter, property);
    type = modifyTypeByAnnotation(config, getter, type, name);
    TypeDeserializer typeDeser = type.getTypeHandler();
    SettableBeanProperty prop = new SettableBeanProperty.SetterlessProperty(name, type, typeDeser,
            beanDesc.getClassAnnotations(), getter);
    if (propDeser != null) {
        prop.setValueDeserializer(propDeser);
    }
    return prop;
```

```
    }

    /*
    /**********************************************************
    /* Helper methods for Bean deserializer, other
    /**********************************************************
     */

    /**
     * Helper method used to skip processing for types that we know
     * can not be (i.e. are never consider to be) beans:
     * things like primitives, Arrays, Enums, and proxy types.
     *<p>
     * Note that usually we shouldn't really be getting these sort of
     * types anyway; but better safe than sorry.
     */
    protected boolean isPotentialBeanType(Class<?> type)
    {
        String typeStr = ClassUtil.canBeABeanType(type);
        if (typeStr != null) {
            throw new IllegalArgumentException("Can not deserialize Class "+type.getName()+" (of type "+typeStr+")
as a Bean");
        }
        if (ClassUtil.isProxyType(type)) {
            throw new IllegalArgumentException("Can not deserialize Proxy class "+type.getName()+" as a Bean");
        }
        // also: can't deserialize local (in-method, anonymous, non-static-enclosed) classes
        typeStr = ClassUtil.isLocalType(type);
        if (typeStr != null) {
            throw new IllegalArgumentException("Can not deserialize Class "+type.getName()+" (of type "+typeStr+")
as a Bean");
        }
       return true;
    }

    /**
     * Helper method that will check whether given raw type is marked as always ignorable
     * (for purpose of ignoring properties with type)
     */
    protected boolean isIgnorableType(DeserializationConfig config, BasicBeanDescription beanDesc,
            Class<?> type, Map<Class<?>,Boolean> ignoredTypes)
    {
        Boolean status = ignoredTypes.get(type);
        if (status == null) {
            BasicBeanDescription desc = config.introspectClassAnnotations(type);
            status = config.getAnnotationIntrospector().isIgnorableType(desc.getClassInfo());
            // We default to 'false', ie. not ignorable
            if (status == null) {
```

```
            status = Boolean.FALSE;
          }
        }
        return status;
    }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;

import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.type.JavaType;

/**
* Deserializer only used for abstract types used as placeholders during polymorphic
* type handling deserialization. If so, there is no real deserializer associated
* with nominal type, just {@link TypeDeserializer}; and any calls that do not
* pass such resolver will result in an error.
*
* @author tatu
*
* @since 1.6
*/
public class AbstractDeserializer
    extends JsonDeserializer<Object>
{
    protected final JavaType _baseType;

    public AbstractDeserializer(JavaType bt)
    {
        _baseType = bt;
    }

    @Override
    public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
            TypeDeserializer typeDeserializer)
        throws IOException, JsonProcessingException
    {
        /* As per [JACKSON-417], there is a chance we might be "native" types (String, Boolean,
         * Integer), which do not include any type information...
         */
        switch (jp.getCurrentToken()) {
        /* First, so-called "native" types (ones that map
         * naturally and thus do not need or use type ids)
         */
        case VALUE_STRING:
```

```
            return jp.getText();

        case VALUE_NUMBER_INT:
            // For [JACKSON-100], see above:
            if (ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_INTEGER_FOR_INTS)) {
                return jp.getBigIntegerValue();
            }
            return jp.getIntValue();

        case VALUE_NUMBER_FLOAT:
            // For [JACKSON-72], see above
            if (ctxt.isEnabled(DeserializationConfig.Feature.USE_BIG_DECIMAL_FOR_FLOATS)) {
                return jp.getDecimalValue();
            }
            return Double.valueOf(jp.getDoubleValue());

        case VALUE_TRUE:
            return Boolean.TRUE;
        case VALUE_FALSE:
            return Boolean.FALSE;
        case VALUE_EMBEDDED_OBJECT:
            return jp.getEmbeddedObject();

        case VALUE_NULL: // should not get this far really but...
            return null;

        case START_ARRAY:
            /* 11-Feb-2011, tatu: Uh. Given that we know very little about the type
             *   here, we can't be sure but it sure looks like we must have used
             *   "As.WRAPPER_ARRAY" here. This will NOT work properly if someone
             *   tries to use "As.WRAPPER_OBJECT"; but let's tackle one issue
             *   at a time. See [JACKSON-485] for an example of where this fix
             *   is needed.
             */
            return typeDeserializer.deserializeTypedFromAny(jp, ctxt);

            /*
        case START_OBJECT:
        case FIELD_NAME:
             */
        }

        // should we call 'fromAny' or 'fromObject'? We should get an object, for abstract types, right?
        // 11-Feb-2011: not necessarily; for example, when serialize Enums that implement an interface... *sigh*
        //return typeDeserializer.deserializeTypedFromAny(jp, ctxt);

        return typeDeserializer.deserializeTypedFromObject(jp, ctxt);
    }
```

```
  @Override
  public Object deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
  {
     // no can do:
     throw ctxt.instantiationException(_baseType.getRawClass(), "abstract types can only be instantiated with
additional type information");
  }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.lang.reflect.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.JsonMappingException;
import org.codehaus.jackson.map.introspect.AnnotatedMethod;
import org.codehaus.jackson.type.JavaType;

/**
* Class that represents a "wildcard" set method which can be used
* to generically set values of otherwise unmapped (aka "unknown")
* properties read from Json content.
*<p>
* !!! Note: might make sense to refactor to share some code
* with {@link SettableBeanProperty}?
*/
public final class SettableAnyProperty
{
  /**
   * Method used for setting "any" properties, along with annotation
   * information. Retained to allow contextualization of any properties.
   *
   * @since 1.7
   */
  final protected BeanProperty _property;

  /**
   * Physical JDK object used for assigning properties.
   */
  final protected Method _setter;

  final protected JavaType _type;
```

```
protected JsonDeserializer<Object> _valueDeserializer;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

public SettableAnyProperty(BeanProperty property, AnnotatedMethod setter, JavaType type)
{
   _property = property;
   _type = type;
   _setter = setter.getAnnotated();
}

public void setValueDeserializer(JsonDeserializer<Object> deser)
{
   if (_valueDeserializer != null) { // sanity check
      throw new IllegalStateException("Already had assigned deserializer for SettableAnyProperty");
   }
   _valueDeserializer = deser;
}

/*
/**********************************************************
/* Public API, accessors
/**********************************************************
 */

public BeanProperty getProperty() { return _property; }

public boolean hasValueDeserializer() { return (_valueDeserializer != null); }

public JavaType getType() { return _type; }

/*
/**********************************************************
/* Public API, deserialization
/**********************************************************
 */

/**
 * Method called to deserialize appropriate value, given parser (and
 * context), and set it using appropriate method (a setter method).
 */
public final void deserializeAndSet(JsonParser jp, DeserializationContext ctxt,
                                    Object instance, String propName)
   throws IOException, JsonProcessingException
```

```
{
   set(instance, propName, deserialize(jp, ctxt));
}

public final Object deserialize(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   JsonToken t = jp.getCurrentToken();
   if (t == JsonToken.VALUE_NULL) {
      return null;
   }
   return _valueDeserializer.deserialize(jp, ctxt);
}

public final void set(Object instance, String propName, Object value)
   throws IOException
{
   try {
      _setter.invoke(instance, propName, value);
   } catch (Exception e) {
      _throwAsIOE(e, propName, value);
   }
}

/*
/**********************************************************
/* Helper methods
/**********************************************************
 */

/**
 * @param e Exception to re-throw or wrap
 * @param propName Name of property (from Json input) to set
 * @param value Value of the property
 */
protected void _throwAsIOE(Exception e, String propName, Object value)
   throws IOException
{
   if (e instanceof IllegalArgumentException) {
      String actType = (value == null) ? "[NULL]" : value.getClass().getName();
      StringBuilder msg = new StringBuilder("Problem deserializing \"any\" property '").append(propName);
      msg.append("' of class "+getClassName()+" (expected type: ").append(_type);
      msg.append("; actual type: ").append(actType).append(")");
      String origMsg = e.getMessage();
      if (origMsg != null) {
         msg.append(", problem: ").append(origMsg);
      } else {
         msg.append(" (no error message provided)");
```

```
            }
            throw new JsonMappingException(msg.toString(), null, e);
        }
        if (e instanceof IOException) {
            throw (IOException) e;
        }
        if (e instanceof RuntimeException) {
            throw (RuntimeException) e;
        }
        // let's wrap the innermost problem
        Throwable t = e;
        while (t.getCause() != null) {
            t = t.getCause();
        }
        throw new JsonMappingException(t.getMessage(), null, t);
    }

    private String getClassName() { return _setter.getDeclaringClass().getName(); }

    @Override public String toString() { return "[any property on class "+getClassName()+"]"; }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;

import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.map.type.ArrayType;
import org.codehaus.jackson.map.util.ObjectBuffer;
import org.codehaus.jackson.type.JavaType;

/**
* Basic serializer that can serialize non-primitive arrays.
*/
@JacksonStdImpl
public class ArrayDeserializer
    extends ContainerDeserializer<Object[]>
{
    // // Configuration

    protected final JavaType _arrayType;

    /**
     * Flag that indicates whether the component type is Object or not.
     * Used for minor optimization when constructing result.
```

```
 */
protected final boolean _untyped;

/**
 * Type of contained elements: needed for constructing actual
 * result array
 */
protected final Class<?> _elementClass;

/**
 * Element deserializer
 */
protected final JsonDeserializer<Object> _elementDeserializer;

/**
 * If element instances have polymorphic type information, this
 * is the type deserializer that can handle it
 */
final TypeDeserializer _elementTypeDeserializer;

@Deprecated
public ArrayDeserializer(ArrayType arrayType, JsonDeserializer<Object> elemDeser)
{
   this(arrayType, elemDeser, null);
}

public ArrayDeserializer(ArrayType arrayType, JsonDeserializer<Object> elemDeser,
       TypeDeserializer elemTypeDeser)
{
   super(Object[].class);
   _arrayType = arrayType;
   _elementClass = arrayType.getContentType().getRawClass();
   _untyped = (_elementClass == Object.class);
   _elementDeserializer = elemDeser;
   _elementTypeDeserializer = elemTypeDeser;
}

/*
/**********************************************************
/* ContainerDeserializer API
/**********************************************************
 */

@Override
public JavaType getContentType() {
   return _arrayType.getContentType();
}
```

```
@Override
public JsonDeserializer<Object> getContentDeserializer() {
   return _elementDeserializer;
}

/*
/**********************************************************
/* JsonDeserializer API
/**********************************************************
 */

@Override
public Object[] deserialize(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   // Ok: must point to START_ARRAY (or equivalent)
   if (!jp.isExpectedStartArrayToken()) {
      /* 04-Oct-2009, tatu: One exception; byte arrays are generally
       *   serialized as base64, so that should be handled
       */
      if (jp.getCurrentToken() == JsonToken.VALUE_STRING
         && _elementClass == Byte.class) {
         return deserializeFromBase64(jp, ctxt);
      }
      throw ctxt.mappingException(_arrayType.getRawClass());
   }

   final ObjectBuffer buffer = ctxt.leaseObjectBuffer();
   Object[] chunk = buffer.resetAndStart();
   int ix = 0;
   JsonToken t;
   final TypeDeserializer typeDeser = _elementTypeDeserializer;

   while ((t = jp.nextToken()) != JsonToken.END_ARRAY) {
      // Note: must handle null explicitly here; value deserializers won't
      Object value;

      if (t == JsonToken.VALUE_NULL) {
         value = null;
      } else if (typeDeser == null) {
         value = _elementDeserializer.deserialize(jp, ctxt);
      } else {
         value = _elementDeserializer.deserializeWithType(jp, ctxt, typeDeser);
      }
      if (ix >= chunk.length) {
         chunk = buffer.appendCompletedChunk(chunk);
         ix = 0;
      }
```

```
            chunk[ix++] = value;
        }

        Object[] result;

        if (_untyped) {
            result = buffer.completeAndClearBuffer(chunk, ix);
        } else {
            result = buffer.completeAndClearBuffer(chunk, ix, _elementClass);
        }
        ctxt.returnObjectBuffer(buffer);
        return result;
    }

    @Override
    public Object[] deserializeWithType(JsonParser jp, DeserializationContext ctxt,
            TypeDeserializer typeDeserializer)
        throws IOException, JsonProcessingException
    {
        /* Should there be separate handling for base64 stuff?
         * for now this should be enough:
         */
        return (Object[]) typeDeserializer.deserializeTypedFromArray(jp, ctxt);
    }

    /*
    /**********************************************************
    /* Internal methods
    /**********************************************************
     */

    protected Byte[] deserializeFromBase64(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        // First same as what ArrayDeserializers.ByteDeser does:
        byte[] b = jp.getBinaryValue(ctxt.getBase64Variant());
        // But then need to convert to wrappers
        Byte[] result = new Byte[b.length];
        for (int i = 0, len = b.length; i < len; ++i) {
            result[i] = Byte.valueOf(b[i]);
        }
        return result;
    }
}
package org.codehaus.jackson.map.deser;

import java.util.*;
```

```
import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.type.*;

/**
* Deserializer factory implementation that allows for configuring
* mapping between types and deserializers to use, by using
* multiple types of overrides. Existing mappings established by
* {@link BeanDeserializerFactory} (and its super class,
* {@link BasicDeserializerFactory}) are used if no overrides are
* defined.
*<p>
* Unlike base deserializer factories, this factory is stateful because
* of configuration settings. It is thread-safe, however, as long as
* all configuration as done before using the factory -- a single
* instance can be shared between providers and mappers.
*<p>
* Configurations currently available are:
*<ul>
* <li>Ability to define explicit mappings between simple non-generic
*   classes/interfaces and deserializers to use for deserializing
*   instance of these classes. Mappings are one-to-one (i.e. there is
*   no "generic" variant for handling sub- or super-classes/interfaces).
*  </li>
*</ul>
*<p>
* The way custom deserializer factory instances are hooked to
* {@link ObjectMapper} is usually by constructing an instance of
* {@link DeserializerProvider} (most commonly
* {@link StdDeserializerProvider}) with custom deserializer factory,
* and setting {@link ObjectMapper} to use it.
*/
public class CustomDeserializerFactory
  extends BeanDeserializerFactory
{
  /*
  /**********************************************************
  /* Configuration, direct/special mappings
  /**********************************************************
   */

  /**
   * Direct mappings that are used for exact class and interface type
   * matches.
   */
  protected HashMap<ClassKey,JsonDeserializer<Object>> _directClassMappings = null;

  /*
```

```
/**********************************************************
/* Configuration: mappings that define "mix-in annotations"
/**********************************************************
 */

/**
 * Mapping that defines how to apply mix-in annotations: key is
 * the type to received additional annotations, and value is the
 * type that has annotations to "mix in".
 *<p>
 * Annotations associated with the value classes will be used to
 * override annotations of the key class, associated with the
 * same field or method. They can be further masked by sub-classes:
 * you can think of it as injecting annotations between the target
 * class and its sub-classes (or interfaces)
 *
 * @since 1.2
 */
protected HashMap<ClassKey,Class<?>> _mixInAnnotations;

/*
/**********************************************************
/* Life-cycle, constructors
/**********************************************************
 */

public CustomDeserializerFactory() {
   this(null);
}

protected CustomDeserializerFactory(Config config)
{
   super(config);
}

@Override
public DeserializerFactory withConfig(Config config)
{
   // See super-class method for reasons for this check...
   if (getClass() != CustomDeserializerFactory.class) {
      throw new IllegalStateException("Subtype of CustomDeserializerFactory ("+getClass().getName()
            +") has not properly overridden method 'withAdditionalDeserializers': can not instantiate subtype with "
            +"additional deserializer definitions");
   }
   return new CustomDeserializerFactory(config);
}

/*
```

```
/**********************************************************
/* Configuration: type-to-serializer mappings
/**********************************************************
 */

/**
 * Method used to add a mapping for specific type -- and only that
 * type -- to use specified deserializer.
 * This means that binding is not used for sub-types.
 *<p>
 * Note that both class and interfaces can be mapped, since the type
 * is derived from method declarations; and hence may be abstract types
 * and interfaces. This is different from custom serialization where
 * only class types can be directly mapped.
 *
 * @param forClass Class to deserialize using specific deserializer.
 * @param deser Deserializer to use for the class. Declared type for
 *   deserializer may be more specific (sub-class) than declared class
 *   to map, since that will still be compatible (deserializer produces
 *   sub-class which is assignable to field/method)
 */
@SuppressWarnings("unchecked")
public <T> void addSpecificMapping(Class<T> forClass, JsonDeserializer<? extends T> deser)
{
   ClassKey key = new ClassKey(forClass);
   if (_directClassMappings == null) {
      _directClassMappings = new HashMap<ClassKey,JsonDeserializer<Object>>();
   }
   _directClassMappings.put(key, (JsonDeserializer<Object>)deser);
}

/**
 * Method to use for adding mix-in annotations that Class
 * <code>classWithMixIns</code> contains into class
 * <code>destinationClass</code>. Mixing in is done when introspecting
 * class annotations and properties.
 * Annotations from <code>classWithMixIns</code> (and its supertypes)
 * will <b>override</b>
 * anything <code>destinationClass</code> (and its super-types)
 * has already.
 *
 * @param destinationClass Class to modify by adding annotations
 * @param classWithMixIns Class that contains annotations to add
 *
 * @since 1.2
 */
public void addMixInAnnotationMapping(Class<?> destinationClass,
                                      Class<?> classWithMixIns)
```

```
{
   if (_mixInAnnotations == null) {
      _mixInAnnotations = new HashMap<ClassKey,Class<?>>();
   }
   _mixInAnnotations.put(new ClassKey(destinationClass), classWithMixIns);
}

/*
/**********************************************************
/* DeserializerFactory API
/**********************************************************
 */

@Override
public JsonDeserializer<Object> createBeanDeserializer(DeserializationConfig config, DeserializerProvider p,
      JavaType type, BeanProperty property)
   throws JsonMappingException
{
   Class<?> cls = type.getRawClass();
   ClassKey key = new ClassKey(cls);

   // Do we have a match?
   if (_directClassMappings != null) {
      JsonDeserializer<Object> deser = _directClassMappings.get(key);
      if (deser != null) {
         return deser;
      }
   }
   // If not, let super class do its job
   return super.createBeanDeserializer(config, p, type, property);
}

@Override
public JsonDeserializer<?> createArrayDeserializer(DeserializationConfig config, DeserializerProvider p,
      ArrayType type, BeanProperty property)
   throws JsonMappingException
{
   ClassKey key = new ClassKey(type.getRawClass());
   if (_directClassMappings != null) {
      JsonDeserializer<Object> deser = _directClassMappings.get(key);
      if (deser != null) {
         return deser;
      }
   }
   return super.createArrayDeserializer(config, p, type, property);
}

//public JsonDeserializer<?> createCollectionDeserializer(...) throws JsonMappingException
```

```
  @Override
  public JsonDeserializer<?> createEnumDeserializer(DeserializationConfig config, DeserializerProvider p,
        JavaType enumType, BeanProperty property)
    throws JsonMappingException
  {
    /* Enums can't extend anything; must be a direct
     * match, if anything:
     * (theoretically they can implement interfaces, but that seems irrelevant)
     */
    if (_directClassMappings != null) {
      ClassKey key = new ClassKey(enumType.getRawClass());
      JsonDeserializer<?> deser = _directClassMappings.get(key);
      if (deser != null) {
        return deser;
      }
    }
    return super.createEnumDeserializer(config, p, enumType, property);
  }

  //public JsonDeserializer<?> createMapDeserializer(...) throws JsonMappingException

  //public JsonDeserializer<?> createTreeDeserializer(...) throws JsonMappingException
}
package org.codehaus.jackson.map.deser;

import org.codehaus.jackson.map.DeserializationConfig;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.deser.BeanDeserializer;
import org.codehaus.jackson.map.deser.BeanDeserializerFactory;
import org.codehaus.jackson.map.introspect.BasicBeanDescription;

/**
* Abstract class that defines API for objects that can be registered (for {@link BeanDeserializerFactory}
* to participate in constructing {@link BeanDeserializer} instances.
* This is typically done by modules that want alter some aspects of deserialization
* process; and is preferable to sub-classing of {@link BeanDeserializerFactory}.
*<p>
* Sequence in which callback methods are called is as follows:
* <ol>
* </ol>
*<p>
* Default method implementations are "no-op"s, meaning that methods are implemented
* but have no effect.
*
* @since 1.7
*/
public abstract class BeanDeserializerModifier
```

```
{
  /**
   * Method called by {@link BeanDeserializerFactory} when it has collected
   * basic information such as tentative list of properties to deserializer.
   *
   * Implementations may choose to modify state of builder (to affect deserializer being
   * built), or even completely replace it (if they want to build different kind of
   * deserializer). Typically changes mostly concern set of properties to deserialize.
   */
  public BeanDeserializerBuilder updateBuilder(DeserializationConfig config,
          BasicBeanDescription beanDesc, BeanDeserializerBuilder builder) {
    return builder;
  }

  /**
   * Method called by {@link BeanDeserializerFactory} after constructing default
   * bean deserializer instance with properties collected and ordered earlier.
   * Implementations can modify or replace given deserializer and return deserializer
   * to use. Note that although initial deserializer being passed is of type
   * {@link BeanDeserializer}, modifiers may return deserializers of other types;
   * and this is why implementations must check for type before casting.
   */
  public JsonDeserializer<?> modifyDeserializer(DeserializationConfig config,
          BasicBeanDescription beanDesc, JsonDeserializer<?> deserializer) {
    return deserializer;
  }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;

import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.io.NumberInput;
import org.codehaus.jackson.map.*;

/**
* Base class for simple key deserializers.
*/
public abstract class StdKeyDeserializer
  extends KeyDeserializer
{
  final protected Class<?> _keyClass;

  protected StdKeyDeserializer(Class<?> cls) { _keyClass = cls; }

  @Override
```

```
public final Object deserializeKey(String key, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   if (key == null) { // is this even legal call?
      return null;
   }
   try {
      Object result = _parse(key, ctxt);
      if (result != null) {
         return result;
      }
   } catch (Exception re) {
      throw ctxt.weirdKeyException(_keyClass, key, "not a valid representation: "+re.getMessage());
   }
   throw ctxt.weirdKeyException(_keyClass, key, "not a valid representation");
}

public Class<?> getKeyClass() { return _keyClass; }

protected abstract Object _parse(String key, DeserializationContext ctxt) throws Exception;

/*
/**********************************************************
/* Helper methods for sub-classes
/**********************************************************
 */

protected int _parseInt(String key) throws IllegalArgumentException
{
   return Integer.parseInt(key);
}

protected long _parseLong(String key) throws IllegalArgumentException
{
   return Long.parseLong(key);
}

protected double _parseDouble(String key) throws IllegalArgumentException
{
   return NumberInput.parseDouble(key);
}

/*
/**********************************************************
/* Key deserializer implementations; wrappers
/**********************************************************
 */
```

```
 final static class BoolKD extends StdKeyDeserializer
 {
    BoolKD() { super(Boolean.class); }

    @Override
    public Boolean _parse(String key, DeserializationContext ctxt) throws JsonMappingException
    {
       if ("true".equals(key)) {
          return Boolean.TRUE;
       }
       if ("false".equals(key)) {
          return Boolean.FALSE;
       }
       throw ctxt.weirdKeyException(_keyClass, key, "value not 'true' or 'false'");
    }
 }

 final static class ByteKD extends StdKeyDeserializer
 {
    ByteKD() { super(Byte.class); }

    @Override
 public Byte _parse(String key, DeserializationContext ctxt) throws JsonMappingException
    {
       int value = _parseInt(key);
       if (value < Byte.MIN_VALUE || value > Byte.MAX_VALUE) {
          throw ctxt.weirdKeyException(_keyClass, key, "overflow, value can not be represented as 8-bit value");
       }
       return Byte.valueOf((byte) value);
    }
 }

 final static class ShortKD extends StdKeyDeserializer
 {
    ShortKD() { super(Integer.class); }

    @Override
 public Short _parse(String key, DeserializationContext ctxt) throws JsonMappingException
    {
       int value = _parseInt(key);
       if (value < Short.MIN_VALUE || value > Short.MAX_VALUE) {
          throw ctxt.weirdKeyException(_keyClass, key, "overflow, value can not be represented as 16-bit value");
       }
       return Short.valueOf((short) value);
    }
 }

 /**
```

```
 * Dealing with Characters is bit trickier: let's assume it must be a String, and that
 * Unicode numeric value is never used.
 */
 final static class CharKD extends StdKeyDeserializer
 {
    CharKD() { super(Character.class); }

    @Override
 public Character _parse(String key, DeserializationContext ctxt) throws JsonMappingException
    {
       if (key.length() == 1) {
          return Character.valueOf(key.charAt(0));
       }
       throw ctxt.weirdKeyException(_keyClass, key, "can only convert 1-character Strings");
    }
 }

 final static class IntKD extends StdKeyDeserializer
 {
    IntKD() { super(Integer.class); }

    @Override
 public Integer _parse(String key, DeserializationContext ctxt) throws JsonMappingException
    {
       return _parseInt(key);
    }
 }

 final static class LongKD extends StdKeyDeserializer
 {
    LongKD() { super(Long.class); }

    @Override
    public Long _parse(String key, DeserializationContext ctxt) throws JsonMappingException
    {
       return _parseLong(key);
    }
 }

 final static class DoubleKD extends StdKeyDeserializer
 {
    DoubleKD() { super(Double.class); }

    @Override
    public Double _parse(String key, DeserializationContext ctxt) throws JsonMappingException
    {
       return _parseDouble(key);
    }
```

```
}

final static class FloatKD extends StdKeyDeserializer
{
   FloatKD() { super(Float.class); }

   @Override
   public Float _parse(String key, DeserializationContext ctxt) throws JsonMappingException
   {
      /* 22-Jan-2009, tatu: Bounds/range checks would be tricky
       *   here, so let's not bother even trying...
       */
      return Float.valueOf((float) _parseDouble(key));
   }
}

/*
/**********************************************************
/* Key deserializer implementations; other
/**********************************************************
 */

final static class EnumKD extends StdKeyDeserializer
{
   final EnumResolver<?> _resolver;

   EnumKD(EnumResolver<?> er)
   {
      super(er.getEnumClass());
      _resolver = er;
   }

   @Override
   public Enum<?> _parse(String key, DeserializationContext ctxt) throws JsonMappingException
   {
      Enum<?> e = _resolver.findEnum(key);
      if (e == null) {
         throw ctxt.weirdKeyException(_keyClass, key, "not one of values for Enum class");
      }
      return e;
   }
}

/**
 * Key deserializer that calls a single-string-arg constructor
 * to instantiate desired key type.
 */
final static class StringCtorKeyDeserializer extends StdKeyDeserializer
```

```
    {
        final Constructor<?> _ctor;

        public StringCtorKeyDeserializer(Constructor<?> ctor) {
            super(ctor.getDeclaringClass());
            _ctor = ctor;
        }

        @Override
        public Object _parse(String key, DeserializationContext ctxt) throws Exception
        {
            return _ctor.newInstance(key);
        }
    }

    /**
     * Key deserializer that calls a static no-args factory method
     * to instantiate desired key type.
     */
    final static class StringFactoryKeyDeserializer extends StdKeyDeserializer
    {
        final Method _factoryMethod;

        public StringFactoryKeyDeserializer(Method fm) {
            super(fm.getDeclaringClass());
            _factoryMethod = fm;
        }

        @Override
        public Object _parse(String key, DeserializationContext ctxt) throws Exception
        {
            return _factoryMethod.invoke(null, key);
        }
    }
}
package org.codehaus.jackson.map.deser;

import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.type.JavaType;

/**
* Intermediate base deserializer class that adds more shared accessor
* so that other classes can access information about contained (value)
* types
*
* @since 1.6
*/
public abstract class ContainerDeserializer<T>
```

```
  extends StdDeserializer<T>
{
  protected ContainerDeserializer(Class<?> selfType)
  {
     super(selfType);
  }

  /*
  /**********************************************************
  /* Extended API
  /**********************************************************
   */

  /**
   * Accessor for declared type of contained value elements; either exact
   * type, or one of its supertypes.
   */
  public abstract JavaType getContentType();

  /**
   * Accesor for deserializer use for deserializing content values.
   */
  public abstract JsonDeserializer<Object> getContentDeserializer();
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.lang.reflect.Constructor;
import java.util.*;

import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.TypeDeserializer;
import org.codehaus.jackson.map.annotate.JacksonStdImpl;
import org.codehaus.jackson.type.JavaType;

/**
* Basic serializer that can take JSON "Array" structure and
* construct a {@link java.util.Collection} instance, with typed contents.
*<p>
* Note: for untyped content (one indicated by passing Object.class
* as the type), {@link UntypedObjectDeserializer} is used instead.
* It can also construct {@link java.util.List}s, but not with specific
* POJO types, only other containers and primitives/wrappers.
*/
```

```
@JacksonStdImpl
public class CollectionDeserializer
  extends ContainerDeserializer<Collection<Object>>
{
  // // Configuration

  protected final JavaType _collectionType;

  /**
   * Value deserializer.
   */
  final JsonDeserializer<Object> _valueDeserializer;

  /**
   * If element instances have polymorphic type information, this
   * is the type deserializer that can handle it
   */
  final TypeDeserializer _valueTypeDeserializer;

  /**
   * We will use the default constructor of the class for
   * instantiation
   */
  final Constructor<Collection<Object>> _defaultCtor;

  public CollectionDeserializer(JavaType collectionType, JsonDeserializer<Object> valueDeser,
        TypeDeserializer valueTypeDeser,
        Constructor<Collection<Object>> ctor)
  {
     super(collectionType.getRawClass());
     _collectionType = collectionType;
     _valueDeserializer = valueDeser;
     _valueTypeDeserializer = valueTypeDeser;
     if (ctor == null) {
        throw new IllegalArgumentException("No default constructor found for container class
"+collectionType.getRawClass().getName());
     }
     _defaultCtor = ctor;
  }

  /*
  /**********************************************************
  /* ContainerDeserializer API
  /**********************************************************
   */

  @Override
  public JavaType getContentType() {
```

```
    return _collectionType.getContentType();
}

@Override
public JsonDeserializer<Object> getContentDeserializer() {
    return _valueDeserializer;
}

/*
/**********************************************************
/* JsonDeserializer API
/**********************************************************
 */

@Override
public Collection<Object> deserialize(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    Collection<Object> result;
    try {
        result = _defaultCtor.newInstance();
    } catch (Exception e) {
        throw ctxt.instantiationException(_collectionType.getRawClass(), e);
    }
    return deserialize(jp, ctxt, result);
}

@Override
public Collection<Object> deserialize(JsonParser jp, DeserializationContext ctxt,
                                      Collection<Object> result)
    throws IOException, JsonProcessingException
{
    // Ok: must point to START_ARRAY (or equivalent)
    if (!jp.isExpectedStartArrayToken()) {
        throw ctxt.mappingException(_collectionType.getRawClass());
    }

    JsonDeserializer<Object> valueDes = _valueDeserializer;
    JsonToken t;
    final TypeDeserializer typeDeser = _valueTypeDeserializer;

    while ((t = jp.nextToken()) != JsonToken.END_ARRAY) {
        Object value;

        if (t == JsonToken.VALUE_NULL) {
            value = null;
        } else if (typeDeser == null) {
            value = valueDes.deserialize(jp, ctxt);
```

```
            } else {
                value = valueDes.deserializeWithType(jp, ctxt, typeDeser);
            }
            result.add(value);
        }
        return result;
    }

    @Override
    public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
            TypeDeserializer typeDeserializer)
        throws IOException, JsonProcessingException
    {
        // In future could check current token... for now this should be enough:
        return typeDeserializer.deserializeTypedFromArray(jp, ctxt);
    }
}
package org.codehaus.jackson.map.deser;

import java.io.IOException;
import java.lang.reflect.*;
import java.util.*;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.annotate.JsonCachable;
import org.codehaus.jackson.map.deser.impl.BeanPropertyMap;
import org.codehaus.jackson.map.introspect.AnnotatedClass;
import org.codehaus.jackson.map.type.ClassKey;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.map.util.ClassUtil;
import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.util.TokenBuffer;

/**
* Deserializer class that can deserialize instances of
* arbitrary bean objects, usually from JSON Object structs,
* but possibly also from simple types like String values.
*/
@JsonCachable
/* Because of costs associated with constructing bean deserializers,
* they usually should be cached unlike other deserializer types.
* But more importantly, it is important to be able to cache
* bean serializers to handle cyclic references.
*/
public class BeanDeserializer
    extends StdDeserializer<Object>
    implements ResolvableDeserializer
```

```
{
 /*
 /**********************************************************
 /* Information regarding type being deserialized
 /**********************************************************
  */

 /**
  * Class for which deserializer is built; used for accessing
  * annotations during resolution phase (see {@link #resolve}).
  */
 final protected AnnotatedClass _forClass;

 /**
  * Declared type of the bean this deserializer handles.
  */
 final protected JavaType _beanType;

 /**
  * Property that contains value to be deserialized using
  * deserializer
  *
  * @since 1.7
  */
 final protected BeanProperty _property;

 /*
 /**********************************************************
 /* Construction configuration
 /**********************************************************
  */

 /**
  * Default constructor used to instantiate the bean when mapping
  * from Json object, and only using setters for initialization
  * (not specific constructors).
  *<p>
  * Note: may be null, if deserializer is constructed for abstract
  * types (which is only useful if additional type information will
  * allow construction of concrete subtype).
  */
 protected final Constructor<?> _defaultConstructor;

 /**
  * If the "bean" class can be instantiated using just a single
  * String (via constructor, static method etc), this object
  * knows how to invoke method/constructor in question.
  * If so, no setters will be used.
```

```
 */
protected final Creator.StringBased _stringCreator;

/**
 * If the "bean" class can be instantiated using just a single
 * numeric (int, long) value  (via constructor, static method etc),
 * this object
 * knows how to invoke method/constructor in question.
 * If so, no setters will be used.
 */
protected final Creator.NumberBased _numberCreator;

/**
 * If the bean class can be instantiated using a creator
 * (an annotated single arg constructor or static method),
 * this object is used for handling details of how delegate-based
 * deserialization and instance construction works
 */
protected final Creator.Delegating _delegatingCreator;

/**
 * If the bean needs to be instantiated using constructor
 * or factory method
 * that takes one or more named properties as argument(s),
 * this creator is used for instantiation.
 */
protected final Creator.PropertyBased _propertyBasedCreator;

/*
/**********************************************************
/* Property information, setters
/**********************************************************
 */

/**
 * Mapping of property names to properties, built when all properties
 * to use have been succesfully resolved.
 *
 * @since 1.7
 */
final protected BeanPropertyMap _beanProperties;

/**
 * Fallback setter used for handling any properties that are not
 * mapped to regular setters. If setter is not null, it will be
 * called once for each such property.
 */
final protected SettableAnyProperty _anySetter;
```

```
/**
 * In addition to properties that are set, we will also keep
 * track of recognized but ignorable properties: these will
 * be skipped without errors or warnings.
 */
final protected HashSet<String> _ignorableProps;

/**
 * Flag that can be set to ignore and skip unknown properties.
 * If set, will not throw an exception for unknown properties.
 */
final protected boolean _ignoreAllUnknown;

/**
 * We may also have one or more back reference fields (usually
 * zero or one).
 */
final protected Map<String, SettableBeanProperty> _backRefs;

/*
/**********************************************************
/* Special deserializers needed for sub-types
/**********************************************************
 */

/**
 * Lazily constructed map used to contain deserializers needed
 * for polymorphic subtypes.
 */
protected HashMap<ClassKey, JsonDeserializer<Object>> _subDeserializers;

/*
/**********************************************************
/* Life-cycle, construction, initialization
/**********************************************************
 */

public BeanDeserializer(AnnotatedClass forClass, JavaType type, BeanProperty property,
      CreatorContainer creators,
      BeanPropertyMap properties, Map<String, SettableBeanProperty> backRefs,
      HashSet<String> ignorableProps, boolean ignoreAllUnknown,
      SettableAnyProperty anySetter)
{
   super(type);
   _forClass = forClass;
   _beanType = type;
   _property = property;
```

```
    _beanProperties = properties;
    _backRefs = backRefs;
    _ignorableProps = ignorableProps;
    _ignoreAllUnknown = ignoreAllUnknown;
    _anySetter = anySetter;

    // And then creator stuff:
    _stringCreator = creators.stringCreator();
    _numberCreator = creators.numberCreator();
    /* Delegating constructor means that the JSON Object is first deserialized
     * into delegated type, and then resulting value is passed as the argument
     * to delegating constructor.
     * Note that delegating constructors have precedence over default
     * and property-based constructors.
     */
    _delegatingCreator = creators.delegatingCreator();
    _propertyBasedCreator = creators.propertyBasedCreator();

    /* important: ensure we do not hold on to default constructor,
     * if delegating OR property-based creator is found
     */
    if (_delegatingCreator != null || _propertyBasedCreator != null) {
        _defaultConstructor = null;
    } else {
        _defaultConstructor = creators.getDefaultConstructor();
    }

}

/**
 * Copy-constructor that can be used by sub-classes to allow
 * copy-on-write styling copying of settings of an existing instance.
 *
 * @since 1.7
 */
protected BeanDeserializer(BeanDeserializer src)
{
    super(src._beanType);
    _forClass = src._forClass;
    _beanType = src._beanType;
    _property = src._property;
    _beanProperties = src._beanProperties;
    _backRefs = src._backRefs;
    _ignorableProps = src._ignorableProps;
    _ignoreAllUnknown = src._ignoreAllUnknown;
    _anySetter = src._anySetter;

    // and plethora of creators...
```

```
    _defaultConstructor = src._defaultConstructor;
    _stringCreator = src._stringCreator;
    _numberCreator = src._numberCreator;
    _delegatingCreator = src._delegatingCreator;
    _propertyBasedCreator = src._propertyBasedCreator;
}

/*
/**********************************************************
/* Public accessors
/**********************************************************
 */

public boolean hasProperty(String propertyName) {
    return _beanProperties.find(propertyName) != null;
}

/**
 * Accessor for checking number of deserialized properties.
 *
 * @since 1.7
 */
public int getPropertyCount() {
    return _beanProperties.size();
}

/*
/**********************************************************
/* Validation, post-processing
/**********************************************************
 */

/**
 * Method called to finalize setup of this deserializer,
 * after deserializer itself has been registered. This
 * is needed to handle recursive and transitive dependencies.
 */
public void resolve(DeserializationConfig config, DeserializerProvider provider)
    throws JsonMappingException
{
    Iterator<SettableBeanProperty> it = _beanProperties.allProperties();
    while (it.hasNext()) {
        SettableBeanProperty prop = it.next();
        // May already have deserializer from annotations, if so, skip:
        if (!prop.hasValueDeserializer()) {
            prop.setValueDeserializer(findDeserializer(config, provider, prop.getType(), prop));
        }
        // and for [JACKSON-235] need to finally link managed references with matching back references
```

```
            String refName = prop.getManagedReferenceName();
            if (refName != null) {
                JsonDeserializer<?> valueDeser = prop._valueDeserializer;
                SettableBeanProperty backProp = null;
                boolean isContainer = false;
                if (valueDeser instanceof BeanDeserializer) {
                    backProp = ((BeanDeserializer) valueDeser).findBackReference(refName);
                } else if (valueDeser instanceof ContainerDeserializer<?>) {
                    JsonDeserializer<?> contentDeser = ((ContainerDeserializer<?>) valueDeser).getContentDeserializer();
                    if (!(contentDeser instanceof BeanDeserializer)) {
                        throw new IllegalArgumentException("Can not handle managed/back reference '"+refName
                                +"': value deserializer is of type ContainerDeserializer, but content type is not handled by a
BeanDeserializer "
                                +" (instead it's of type "+contentDeser.getClass().getName()+")");
                    }
                    backProp = ((BeanDeserializer) contentDeser).findBackReference(refName);
                    isContainer = true;
                } else if (valueDeser instanceof AbstractDeserializer) { // [JACKSON-368]: not easy to fix, alas
                    throw new IllegalArgumentException("Can not handle managed/back reference for abstract types
(property "+_beanType.getRawClass().getName()+"."+prop.getName()+")");
                } else {
                    throw new IllegalArgumentException("Can not handle managed/back reference '"+refName
                            +"': type for value deserializer is not BeanDeserializer or ContainerDeserializer, but "
                            +valueDeser.getClass().getName());
                }
                if (backProp == null) {
                    throw new IllegalArgumentException("Can not handle managed/back reference '"+refName+"': no back
reference property found from type "
                            +prop.getType());
                }
                // also: verify that type is compatible
                JavaType referredType = _beanType;
                JavaType backRefType = backProp.getType();
                if (!backRefType.getRawClass().isAssignableFrom(referredType.getRawClass())) {
                    throw new IllegalArgumentException("Can not handle managed/back reference '"+refName+"': back
reference type ("
                            +backRefType.getRawClass().getName()+") not compatible with managed type ("
                            +referredType.getRawClass().getName()+")");
                }
                _beanProperties.replace(new SettableBeanProperty.ManagedReferenceProperty(refName, prop, backProp,
                        _forClass.getAnnotations(), isContainer));
            }
        }

        // Finally, "any setter" may also need to be resolved now
        if (_anySetter != null && !_anySetter.hasValueDeserializer()) {
            _anySetter.setValueDeserializer(findDeserializer(config, provider, _anySetter.getType(),
_anySetter.getProperty()));
```

```
        }

        // as well as delegate-based constructor:
        if (_delegatingCreator != null) {
            // Need to create a temporary property to allow contextual deserializers:
            BeanProperty.Std property = new BeanProperty.Std(null, _delegatingCreator.getValueType(),
                    _forClass.getAnnotations(), _delegatingCreator.getCreator());
            JsonDeserializer<Object> deser = findDeserializer(config, provider, _delegatingCreator.getValueType(),
property);
            _delegatingCreator.setDeserializer(deser);
        }
        // or property-based one
        if (_propertyBasedCreator != null) {
            for (SettableBeanProperty prop : _propertyBasedCreator.properties()) {
                if (!prop.hasValueDeserializer()) {
                    prop.setValueDeserializer(findDeserializer(config, provider, prop.getType(), prop));
                }
            }
        }
    }
    /*
    /**********************************************************
    /* JsonDeserializer implementation
    /**********************************************************
     */

    /**
     * Main deserialization method for bean-based objects (POJOs).
     */
    @Override
    public final Object deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException, JsonProcessingException
    {
        JsonToken t = jp.getCurrentToken();
        // common case first:
        if (t == JsonToken.START_OBJECT) {
            jp.nextToken();
            return deserializeFromObject(jp, ctxt);
        }
        // and then others, generally requiring use of @JsonCreator
        switch (t) {
        case VALUE_STRING:
      return deserializeFromString(jp, ctxt);
        case VALUE_NUMBER_INT:
        case VALUE_NUMBER_FLOAT:
      return deserializeFromNumber(jp, ctxt);
        case VALUE_EMBEDDED_OBJECT:
            return jp.getEmbeddedObject();
```

```
    case VALUE_TRUE:
    case VALUE_FALSE:
    case START_ARRAY:
        // these only work if there's a (delegating) creator...
        return deserializeUsingCreator(jp, ctxt);
    case FIELD_NAME:
    case END_OBJECT: // added to resolve [JACKSON-319], possible related issues
        return deserializeFromObject(jp, ctxt);
}
    throw ctxt.mappingException(getBeanClass());
 }

 /**
  * Secondary deserialization method, called in cases where POJO
  * instance is created as part of deserialization, potentially
  * after collecting some or all of the properties to set.
  */
 @Override
 public Object deserialize(JsonParser jp, DeserializationContext ctxt, Object bean)
    throws IOException, JsonProcessingException
 {
    JsonToken t = jp.getCurrentToken();
    // 23-Mar-2010, tatu: In some cases, we start with full JSON object too...
    if (t == JsonToken.START_OBJECT) {
        t = jp.nextToken();
    }
    for (; t == JsonToken.FIELD_NAME; t = jp.nextToken()) {
        String propName = jp.getCurrentName();
        SettableBeanProperty prop = _beanProperties.find(propName);
        jp.nextToken(); // skip field, returns value token

        if (prop != null) { // normal case
            try {
                prop.deserializeAndSet(jp, ctxt, bean);
            } catch (Exception e) {
                wrapAndThrow(e, bean, propName, ctxt);
            }
            continue;
        }
        /* As per [JACKSON-313], things marked as ignorable should not be
         * passed to any setter
         */
        if (_ignorableProps != null && _ignorableProps.contains(propName)) {
            jp.skipChildren();
            continue;
        }
        if (_anySetter != null) {
            _anySetter.deserializeAndSet(jp, ctxt, bean, propName);
```

```
            continue;
        }
        // Unknown: let's call handler method
        handleUnknownProperty(jp, ctxt, bean, propName);
    }
    return bean;
}

@Override
public Object deserializeWithType(JsonParser jp, DeserializationContext ctxt,
        TypeDeserializer typeDeserializer)
    throws IOException, JsonProcessingException
{
    // In future could check current token... for now this should be enough:
    return typeDeserializer.deserializeTypedFromObject(jp, ctxt);
}

/*
/**********************************************************
/* Other public accessors
/**********************************************************
 */

public final Class<?> getBeanClass() { return _beanType.getRawClass(); }

@Override public JavaType getValueType() { return _beanType; }

/**
 *
 * @since 1.6
 */
public Iterator<SettableBeanProperty> properties()
{
    if (_beanProperties == null) { // since 1.7
        throw new IllegalStateException("Can only call before BeanDeserializer has been resolved");
    }
    return _beanProperties.allProperties();
}

/**
 * Method needed by {@link BeanDeserializerFactory} to properly link
 * managed- and back-reference pairs.
 */
public SettableBeanProperty findBackReference(String logicalName)
{
    if (_backRefs == null) {
        return null;
    }
```

```
    return _backRefs.get(logicalName);
  }

  /*
  /**********************************************************
  /* Concrete deserialization methods
  /**********************************************************
   */

  public Object deserializeFromObject(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
    if (_defaultConstructor == null) {
      // 25-Jul-2009, tatu: finally, can also use "non-default" constructor (or factory method)
      if (_propertyBasedCreator != null) {
        return _deserializeUsingPropertyBased(jp, ctxt);
      }
     // 07-Jul-2009, tatu: let's allow delegate-based approach too
     if (_delegatingCreator != null) {
       return _delegatingCreator.deserialize(jp, ctxt);
     }
     // should only occur for abstract types...
     if (_beanType.isAbstract()) {
         throw JsonMappingException.from(jp, "Can not instantiate abstract type "+_beanType
             +" (need to add/enable type information?)");
     }
      throw JsonMappingException.from(jp, "No suitable constructor found for type "+_beanType+": can not
instantiate from JSON object (need to add/enable type information?)");
    }

    final Object bean = constructDefaultInstance();
    for (; jp.getCurrentToken() != JsonToken.END_OBJECT; jp.nextToken()) {
      String propName = jp.getCurrentName();
      // Skip field name:
      jp.nextToken();
      SettableBeanProperty prop = _beanProperties.find(propName);
      if (prop != null) { // normal case
        try {
          prop.deserializeAndSet(jp, ctxt, bean);
        } catch (Exception e) {
          wrapAndThrow(e, bean, propName, ctxt);
        }
        continue;
      }
      /* As per [JACKSON-313], things marked as ignorable should not be
       * passed to any setter
       */
      if (_ignorableProps != null && _ignorableProps.contains(propName)) {
```

```
            jp.skipChildren();
        } else if (_anySetter != null) {
            try {
                _anySetter.deserializeAndSet(jp, ctxt, bean, propName);
            } catch (Exception e) {
                wrapAndThrow(e, bean, propName, ctxt);
            }
            continue;
        } else {
            // Unknown: let's call handler method
            handleUnknownProperty(jp, ctxt, bean, propName);
        }
    }
    return bean;
  }

  public Object deserializeFromString(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
   if (_stringCreator != null) {
      return _stringCreator.construct(jp.getText());
    }
   if (_delegatingCreator != null) {
      return _delegatingCreator.deserialize(jp, ctxt);
   }
    throw ctxt.instantiationException(getBeanClass(), "no suitable creator method found to deserialize from JSON
String");
  }

  public Object deserializeFromNumber(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
   if (_numberCreator != null) {
        switch (jp.getNumberType()) {
        case INT:
 return _numberCreator.construct(jp.getIntValue());
        case LONG:
 return _numberCreator.construct(jp.getLongValue());
        }
    }
   if (_delegatingCreator != null) {
      return _delegatingCreator.deserialize(jp, ctxt);
   }
    throw ctxt.instantiationException(getBeanClass(), "no suitable creator method found to deserialize from JSON
Number");
  }

  public Object deserializeUsingCreator(JsonParser jp, DeserializationContext ctxt)
```

```
    throws IOException, JsonProcessingException
{
 if (_delegatingCreator != null) {
     try {
         return _delegatingCreator.deserialize(jp, ctxt);
        } catch (Exception e) {
            wrapAndThrow(e, _beanType.getRawClass(), null, ctxt);
        }
 }
 throw ctxt.mappingException(getBeanClass());
}

/**
 * Method called to deserialize bean using "property-based creator":
 * this means that a non-default constructor or factory method is
 * called, and then possibly other setters. The trick is that
 * values for creator method need to be buffered, first; and
 * due to non-guaranteed ordering possibly some other properties
 * as well.
 *
 * @since 1.2
 */
protected final Object _deserializeUsingPropertyBased(final JsonParser jp, final DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    final Creator.PropertyBased creator = _propertyBasedCreator;
    PropertyValueBuffer buffer = creator.startBuilding(jp, ctxt);

    // 04-Jan-2010, tatu: May need to collect unknown properties for polymorphic cases
    TokenBuffer unknown = null;

    JsonToken t = jp.getCurrentToken();
    for (; t == JsonToken.FIELD_NAME; t = jp.nextToken()) {
        String propName = jp.getCurrentName();
        jp.nextToken(); // to point to value
        // creator property?
        SettableBeanProperty prop = creator.findCreatorProperty(propName);
        if (prop != null) {
            // Last creator property to set?
            Object value = prop.deserialize(jp, ctxt);
            if (buffer.assignParameter(prop.getCreatorIndex(), value)) {
                jp.nextToken(); // to move to following FIELD_NAME/END_OBJECT
                Object bean;
                try {
                    bean = creator.build(buffer);
                } catch (Exception e) {
                    wrapAndThrow(e, _beanType.getRawClass(), propName, ctxt);
                    continue; // never gets here
```

```
            }
          //  polymorphic?
  if (bean.getClass() != _beanType.getRawClass()) {
return handlePolymorphic(jp, ctxt, bean, unknown);
  }
            if (unknown != null) { // nope, just extra unknown stuff...
               bean = handleUnknownProperties(ctxt, bean, unknown);
  }
  // or just clean?
            return deserialize(jp, ctxt, bean);
          }
          continue;
       }
       // regular property? needs buffering
       prop = _beanProperties.find(propName);
       if (prop != null) {
          buffer.bufferProperty(prop, prop.deserialize(jp, ctxt));
          continue;
       }
       /* As per [JACKSON-313], things marked as ignorable should not be
        * passed to any setter
        */
       if (_ignorableProps != null && _ignorableProps.contains(propName)) {
          jp.skipChildren();
          continue;
       }
       // "any property"?
       if (_anySetter != null) {
          buffer.bufferAnyProperty(_anySetter, propName, _anySetter.deserialize(jp, ctxt));
          continue;
       }
       // Ok then, let's collect the whole field; name and value
       if (unknown == null) {
          unknown = new TokenBuffer(jp.getCodec());
       }
       unknown.writeFieldName(propName);
       unknown.copyCurrentStructure(jp);
    }

    // We hit END_OBJECT, so:
    Object bean;
    try {
       bean =  creator.build(buffer);
    } catch (Exception e) {
       wrapAndThrow(e, _beanType.getRawClass(), null, ctxt);
       return null; // never gets here
    }
    if (unknown != null) {
```

```
            // polymorphic?
            if (bean.getClass() != _beanType.getRawClass()) {
                return handlePolymorphic(null, ctxt, bean, unknown);
            }
            // no, just some extra unknown properties
            return handleUnknownProperties(ctxt, bean, unknown);
        }
        return bean;
    }

    /*
    /**********************************************************
    /* Overridable helper methods
    /**********************************************************
     */

    /**
     * Method called when a JSON property is encountered that has not matching
     * setter, any-setter or field, and thus can not be assigned.
     */
    @Override
    protected void handleUnknownProperty(JsonParser jp, DeserializationContext ctxt, Object beanOrClass, String
propName)
        throws IOException, JsonProcessingException
    {
        /* 22-Aug-2010, tatu: Caller now mostly checks for ignorable properties, so
         *    following should not be necessary. However, "handleUnknownProperties()" seems
         *    to still possibly need it so it is left for now.
         */
        // If registered as ignorable, skip
        if (_ignoreAllUnknown ||
            (_ignorableProps != null && _ignorableProps.contains(propName))) {
            jp.skipChildren();
            return;
        }
        /* Otherwise use default handling (call handler(s); if not
         * handled, throw exception or skip depending on settings)
         */
        super.handleUnknownProperty(jp, ctxt, beanOrClass, propName);
    }

    /**
     * Method called to handle set of one or more unknown properties,
     * stored in their entirety in given {@link TokenBuffer}
     * (as field entries, name and value).
     */
    protected Object handleUnknownProperties(DeserializationContext ctxt, Object bean, TokenBuffer
unknownTokens)
```

```
        throws IOException, JsonProcessingException
    {
        // First: add closing END_OBJECT as marker
        unknownTokens.writeEndObject();

        // note: buffer does NOT have starting START_OBJECT
        JsonParser bufferParser = unknownTokens.asParser();
        while (bufferParser.nextToken() != JsonToken.END_OBJECT) {
            String propName = bufferParser.getCurrentName();
            // Unknown: let's call handler method
            bufferParser.nextToken();
            handleUnknownProperty(bufferParser, ctxt, bean, propName);
        }
        return bean;
    }

    /**
     * Method called in cases where we may have polymorphic deserialization
     * case: that is, type of Creator-constructed bean is not the type
     * of deserializer itself. It should be a sub-class or implementation
     * class; either way, we may have more specific deserializer to use
     * for handling it.
     *
     * @param jp (optional) If not null, parser that has more properties to handle
     *   (in addition to buffered properties); if null, all properties are passed
     *   in buffer
     */
    protected Object handlePolymorphic(JsonParser jp, DeserializationContext ctxt,
            Object bean, TokenBuffer unknownTokens)
        throws IOException, JsonProcessingException
    {
        // First things first: maybe there is a more specific deserializer available?
JsonDeserializer<Object> subDeser = _findSubclassDeserializer(ctxt, bean, unknownTokens);
if (subDeser != null) {
    if (unknownTokens != null) {
 // need to add END_OBJECT marker first
 unknownTokens.writeEndObject();
            JsonParser p2 = unknownTokens.asParser();
            p2.nextToken(); // to get to first data field
 bean = subDeser.deserialize(p2, ctxt, bean);
    }
    // Original parser may also have some leftovers
    if (jp != null) {
 bean = subDeser.deserialize(jp, ctxt, bean);
    }
    return bean;
}
// nope; need to use this deserializer. Unknowns we've seen so far?
```

```
if (unknownTokens != null) {
    bean = handleUnknownProperties(ctxt, bean, unknownTokens);
}
// and/or things left to process via main parser?
if (jp != null) {
    bean = deserialize(jp, ctxt, bean);
}
return bean;
  }

  /**
   * Helper method called to (try to) locate deserializer for given sub-type of
   * type that this deserializer handles.
   */
  protected JsonDeserializer<Object> _findSubclassDeserializer(DeserializationContext ctxt, Object bean,
TokenBuffer unknownTokens)
      throws IOException, JsonProcessingException
  {
      JsonDeserializer<Object> subDeser;

      // First: maybe we have already created sub-type deserializer?
      synchronized (this) {
          subDeser = (_subDeserializers == null) ? null : _subDeserializers.get(new ClassKey(bean.getClass()));
      }
      if (subDeser != null) {
          return subDeser;
      }
      // If not, maybe we can locate one. First, need provider
      DeserializerProvider deserProv = ctxt.getDeserializerProvider();
      if (deserProv != null) {
          JavaType type = TypeFactory.type(bean.getClass());
          /* 09-Dec-2010, tatu: Would be nice to know which property pointed to this
           *    bean... but, alas, no such information is retained, so:
           */
          subDeser = deserProv.findValueDeserializer(ctxt.getConfig(), type, _property);
          // Also, need to cache it
          if (subDeser != null) {
              synchronized (this) {
                  if (_subDeserializers == null) {
                      _subDeserializers = new HashMap<ClassKey,JsonDeserializer<Object>>();;
                  }
                  _subDeserializers.put(new ClassKey(bean.getClass()), subDeser);
              }
          }
      }
      return subDeser;
  }
```

```
/**
 * Method that is called to instantiate Object of type this deserializer
 * produces, using the default (no-argument) constructor.
 *
 * @return Instance of type this deserializer handles
 *
 * @since 1.7
 */
protected Object constructDefaultInstance()
{
   try {
      return _defaultConstructor.newInstance();
   } catch (Exception e) {
      ClassUtil.unwrapAndThrowAsIAE(e);
      return null; // never gets here
   }
}

/*
/**********************************************************
/* Helper methods for error reporting
/**********************************************************
 */

/**
 * Method that will modify caught exception (passed in as argument)
 * as necessary to include reference information, and to ensure it
 * is a subtype of {@link IOException}, or an unchecked exception.
 *<p>
 * Rules for wrapping and unwrapping are bit complicated; essentially:
 *<ul>
 * <li>Errors are to be passed as is (if uncovered via unwrapping)
 * <li>"Plain" IOExceptions (ones that are not of type
 *   {@link JsonMappingException} are to be passed as is
 *</ul>
 */
public void wrapAndThrow(Throwable t, Object bean, String fieldName,
      DeserializationContext ctxt)
   throws IOException
{
   /* 05-Mar-2009, tatu: But one nasty edge is when we get
    *   StackOverflow: usually due to infinite loop. But that
    *   usually gets hidden within an InvocationTargetException...
    */
   while (t instanceof InvocationTargetException && t.getCause() != null) {
      t = t.getCause();
   }
   // Errors and "plain" IOExceptions to be passed as is
```

```
    if (t instanceof Error) {
        throw (Error) t;
    }
    boolean wrap = (ctxt == null) || ctxt.isEnabled(DeserializationConfig.Feature.WRAP_EXCEPTIONS);
    // Ditto for IOExceptions; except we may want to wrap mapping exceptions
    if (t instanceof IOException) {
        if (!wrap || !(t instanceof JsonMappingException)) {
            throw (IOException) t;
        }
    } else if (!wrap) { // [JACKSON-407] -- allow disabling wrapping for unchecked exceptions
        if (t instanceof RuntimeException) {
            throw (RuntimeException) t;
        }
    }
    // [JACKSON-55] Need to add reference information
    throw JsonMappingException.wrapWithPath(t, bean, fieldName);
}

public void wrapAndThrow(Throwable t, Object bean, int index, DeserializationContext ctxt)
    throws IOException
{
    while (t instanceof InvocationTargetException && t.getCause() != null) {
        t = t.getCause();
    }
    // Errors and "plain" IOExceptions to be passed as is
    if (t instanceof Error) {
        throw (Error) t;
    }
    boolean wrap = (ctxt == null) || ctxt.isEnabled(DeserializationConfig.Feature.WRAP_EXCEPTIONS);
    // Ditto for IOExceptions; except we may want to wrap mapping exceptions
    if (t instanceof IOException) {
        if (!wrap || !(t instanceof JsonMappingException)) {
            throw (IOException) t;
        }
    } else if (!wrap) { // [JACKSON-407] -- allow disabling wrapping for unchecked exceptions
        if (t instanceof RuntimeException) {
            throw (RuntimeException) t;
        }
    }
    // [JACKSON-55] Need to add reference information
    throw JsonMappingException.wrapWithPath(t, bean, index);
}

/**
 * @deprecated Since 1.7 use variant that takes {@link DeserializationContext}
 */
@Deprecated
public void wrapAndThrow(Throwable t, Object bean, String fieldName)
```

```
        throws IOException
    {
        wrapAndThrow(t, bean, fieldName, null);
    }

    /**
     * @deprecated Since 1.7 use variant that takes {@link DeserializationContext}
     */
    @Deprecated
    public void wrapAndThrow(Throwable t, Object bean, int index)
        throws IOException
    {
        wrapAndThrow(t, bean, index, null);
    }
}
package org.codehaus.jackson.map;

import java.io.*;
import java.net.URL;
import java.util.Collection;
import java.util.concurrent.ConcurrentHashMap;

import org.codehaus.jackson.*;
import org.codehaus.jackson.annotate.JsonTypeInfo;
import org.codehaus.jackson.io.SegmentedStringWriter;
import org.codehaus.jackson.map.deser.BeanDeserializerModifier;
import org.codehaus.jackson.map.deser.StdDeserializationContext;
import org.codehaus.jackson.map.deser.StdDeserializerProvider;
import org.codehaus.jackson.map.introspect.BasicClassIntrospector;
import org.codehaus.jackson.map.introspect.JacksonAnnotationIntrospector;
import org.codehaus.jackson.map.introspect.VisibilityChecker;
import org.codehaus.jackson.map.ser.BeanSerializerModifier;
import org.codehaus.jackson.map.ser.FilterProvider;
import org.codehaus.jackson.map.ser.StdSerializerProvider;
import org.codehaus.jackson.map.ser.BeanSerializerFactory;
import org.codehaus.jackson.map.jsontype.NamedType;
import org.codehaus.jackson.map.jsontype.SubtypeResolver;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.map.jsontype.impl.StdSubtypeResolver;
import org.codehaus.jackson.map.jsontype.impl.StdTypeResolverBuilder;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.node.ArrayNode;
import org.codehaus.jackson.node.JsonNodeFactory;
import org.codehaus.jackson.node.TreeTraversingParser;
import org.codehaus.jackson.node.NullNode;
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.schema.JsonSchema;
import org.codehaus.jackson.type.JavaType;
```

```
import org.codehaus.jackson.type.TypeReference;
import org.codehaus.jackson.util.ByteArrayBuilder;
import org.codehaus.jackson.util.DefaultPrettyPrinter;
import org.codehaus.jackson.util.TokenBuffer;
import org.codehaus.jackson.util.VersionUtil;

/**
* This mapper (or, data binder, or codec) provides functionality for
* converting between Java objects (instances of JDK provided core classes,
* beans), and matching JSON constructs.
* It will use instances of {@link JsonParser} and {@link JsonGenerator}
* for implementing actual reading/writing of JSON.
*<p>
* The main conversion API is defined in {@link ObjectCodec}, so that
* implementation details of this class need not be exposed to
* streaming parser and generator classes.
*<p>
* Note on caching: root-level deserializers are always cached, and accessed
* using full (generics-aware) type information. This is different from
* caching of referenced types, which is more limited and is done only
* for a subset of all deserializer types. The main reason for difference
* is that at root-level there is no incoming reference (and hence no
* referencing property, no referral information or annotations to
* produce differing deserializers), and that the performance impact
* greatest at root level (since it'll essentially cache the full
* graph of deserializers involved).
*/
public class ObjectMapper
  extends ObjectCodec
  implements Versioned
{
  /*
  /**********************************************************
  /* Helper classes, enums
  /**********************************************************
   */

  /**
   * Enumeration used with {@link ObjectMapper#enableDefaultTyping()}
   * to specify what kind of types (classes) default typing should
   * be used for. It will only be used if no explicit type information
   * is found, but this enumeration further limits subset of those
   * types.
   *
   * @since 1.5
   */
  public enum DefaultTyping {
     /**
```

```
     * This value means that only properties that have
     * {@link java.lang.Object} as declared type (including
     * generic types without explicit type) will use default
     * typing.
     */
    JAVA_LANG_OBJECT,

    /**
     * Value that means that default typing will be used for
     * properties with declared type of {@link java.lang.Object}
     * or an abstract type (abstract class or interface).
     * Note that this does <b>not</b> include array types.
     */
    OBJECT_AND_NON_CONCRETE,

    /**
     * Value that means that default typing will be used for
     * all types covered by {@link #OBJECT_AND_NON_CONCRETE}
     * plus all array types for them.
     */
    NON_CONCRETE_AND_ARRAYS,

    /**
     * Value that means that default typing will be used for
     * all non-final types, with exception of small number of
     * "natural" types (String, Boolean, Integer, Double), which
     * can be correctly inferred from JSON; as well as for
     * all arrays of non-final types.
     */
    NON_FINAL
}

/**
 * Customized {@link TypeResolverBuilder} that provides
 * resolver builders based on its configuration. It is used
 * when default typing is enabled (see
 * {@link ObjectMapper#enableDefaultTyping()} for details).
 */
public static class DefaultTypeResolverBuilder
    extends StdTypeResolverBuilder
{
    /**
     * Definition of what types is this default typer valid for.
     */
    protected final DefaultTyping _appliesFor;

    public DefaultTypeResolverBuilder(DefaultTyping t) {
        _appliesFor = t;
```

```
    }

    @Override
    public TypeDeserializer buildTypeDeserializer(JavaType baseType, Collection<NamedType> subtypes,
            BeanProperty property)
    {
        return useForType(baseType) ? super.buildTypeDeserializer(baseType, subtypes, property) : null;
    }

    @Override
    public TypeSerializer buildTypeSerializer(JavaType baseType, Collection<NamedType> subtypes,
            BeanProperty property)
    {
        return useForType(baseType) ? super.buildTypeSerializer(baseType, subtypes, property) : null;
    }

    /**
     * Method called to check if the default type handler should be
     * used for given type.
     * Note: "natural types" (String, Boolean, Integer, Double) will never
     * use typing; that is both due to them being concrete and final,
     * and since actual serializers and deserializers will also ignore any
     * attempts to enforce typing.
     */
    public boolean useForType(JavaType t)
    {
        switch (_appliesFor) {
        case NON_CONCRETE_AND_ARRAYS:
            if (t.isArrayType()) {
                t = t.getContentType();
            }
            // fall through
        case OBJECT_AND_NON_CONCRETE:
            return (t.getRawClass() == Object.class) || !t.isConcrete();
        case NON_FINAL:
            if (t.isArrayType()) {
                t = t.getContentType();
            }
            return !t.isFinal(); // includes Object.class
        default:
        //case JAVA_LANG_OBJECT:
            return (t.getRawClass() == Object.class);
        }
    }
}

/*
/**********************************************************
```

```
  /* Internal constants, singletons
  /**********************************************************
   */

  private final static JavaType JSON_NODE_TYPE = TypeFactory.type(JsonNode.class);

  /* !!! 03-Apr-2009, tatu: Should try to avoid direct reference... but not
   *   sure what'd be simple and elegant way. So until then:
   */
  protected final static ClassIntrospector<? extends BeanDescription> DEFAULT_INTROSPECTOR =
BasicClassIntrospector.instance;

  // 16-May-2009, tatu: Ditto ^^^
  protected final static AnnotationIntrospector DEFAULT_ANNOTATION_INTROSPECTOR = new
JacksonAnnotationIntrospector();

  // @since 1.5
  protected final static VisibilityChecker<?> STD_VISIBILITY_CHECKER =
VisibilityChecker.Std.defaultInstance();

  /*
  /**********************************************************
  /* Configuration settings, shared
  /**********************************************************
   */

  /**
   * Factory used to create {@link JsonParser} and {@link JsonGenerator}
   * instances as necessary.
   */
  protected final JsonFactory _jsonFactory;

  /**
   * Object that defines how to add type information for types that do not
   * have explicit type information settings (which are usually
   * indicated by {@link org.codehaus.jackson.annotate.JsonTypeInfo})
   * If set to null, no type information will be added unless annotations
   * are used; if set to non-null, resolver builder is used to check
   * which type serializers and deserializers are to be used (if any)
   *
   * @since 1.5
   */
  protected TypeResolverBuilder<?> _defaultTyper;

  /**
   * Object used for determining whether specific property elements
   * (method, constructors, fields) can be auto-detected based on
   * their visibility (access modifiers). Can be changed to allow
```

```
 * different minimum visibility levels for auto-detection. Note
 * that this is the global handler; individual types (classes)
 * can further override active checker used (using
 * {@link org.codehaus.jackson.annotate.JsonAutoDetect} annotation)
 *
 * @since 1.5
 */
protected VisibilityChecker<?> _visibilityChecker;

/**
 * Registered concrete subtypes that can be used instead of (or
 * in addition to) ones declared using annotations.
 *
 * @since 1.6
 */
protected SubtypeResolver _subtypeResolver;

/**
 * Custom class loader (or at least one different from loader that loaded
 * Jackson classes) may be needed on some platforms or containers; if so,
 * it is to be defined here.
 *
 * @since 1.6
 */
protected ClassLoader _valueClassLoader;

/*
/**********************************************************
/* Configuration settings, serialization
/**********************************************************
 */

/**
 * Configuration object that defines basic global
 * settings for the serialization process
 */
protected SerializationConfig _serializationConfig;

/**
 * Object that manages access to serializers used for serialization,
 * including caching.
 * It is configured with {@link #_serializerFactory} to allow
 * for constructing custom serializers.
 */
protected SerializerProvider _serializerProvider;

/**
 * Serializer factory used for constructing serializers.
```

```
 */
protected SerializerFactory _serializerFactory;

/*
/**********************************************************
/* Configuration settings, deserialization
/**********************************************************
 */

/**
 * Configuration object that defines basic global
 * settings for the serialization process
 */
protected DeserializationConfig _deserializationConfig;

/**
 * Object that manages access to deserializers used for deserializing
 * JSON content into Java objects, including possible caching
 * of the deserializers. It contains a reference to
 * {@link DeserializerFactory} to use for constructing acutal deserializers.
 */
protected DeserializerProvider _deserializerProvider;

/*
/**********************************************************
/* Caching
/**********************************************************
 */

/* Note: handling of serializers and deserializers is not symmetric;
 * and as a result, only root-level deserializers can be cached here.
 * This is mostly because typing and resolution for deserializers is
 * fully static; whereas it is quite dynamic for serialization.
 */

/**
 * We will use a separate main-level Map for keeping track
 * of root-level deserializers. This is where most succesful
 * cache lookups get resolved.
 * Map will contain resolvers for all kinds of types, including
 * container types: this is different from the component cache
 * which will only cache bean deserializers.
 *<p>
 * Given that we don't expect much concurrency for additions
 * (should very quickly converge to zero after startup), let's
 * explicitly define a low concurrency setting.
 *<p>
 * Since version 1.5, these may are either "raw" deserializers (when
```

```
 * no type information is needed for base type), or type-wrapped
 * deserializers (if it is needed)
 */
final protected ConcurrentHashMap<JavaType, JsonDeserializer<Object>> _rootDeserializers
   = new ConcurrentHashMap<JavaType, JsonDeserializer<Object>>(64, 0.6f, 2);

/*
/**********************************************************
/* Life-cycle (construction, configuration)
/**********************************************************
 */

/**
 * Default constructor, which will construct the default
 * {@link JsonFactory} as necessary, use
 * {@link StdSerializerProvider} as its
 * {@link SerializerProvider}, and
 * {@link BeanSerializerFactory} as its
 * {@link SerializerFactory}.
 * This means that it
 * can serialize all standard JDK types, as well as regular
 * Java Beans (based on method names and Jackson-specific annotations),
 * but does not support JAXB annotations.
 */
public ObjectMapper()
{
   this(null, null, null);
}

/**
 * Construct mapper that uses specified {@link JsonFactory}
 * for constructing necessary {@link JsonParser}s and/or
 * {@link JsonGenerator}s.
 */
public ObjectMapper(JsonFactory jf)
{
   this(jf, null, null);
}

/**
 * Construct mapper that uses specified {@link SerializerFactory}
 * for constructing necessary serializers.
 *
 * @deprecated Use other constructors instead; note that
 *   you can just set serializer factory with {@link #setSerializerFactory}
 */
@Deprecated
public ObjectMapper(SerializerFactory sf)
```

```
    {
        this(null, null, null);
        setSerializerFactory(sf);
    }

    public ObjectMapper(JsonFactory jf,
                        SerializerProvider sp, DeserializerProvider dp)
    {
     this(jf, sp, dp, null, null);
    }

    /**
     *
     * @param jf JsonFactory to use: if null, a new {@link MappingJsonFactory} will be constructed
     * @param sp SerializerProvider to use: if null, a {@link StdSerializerProvider} will be constructed
     * @param dp DeserializerProvider to use: if null, a {@link StdDeserializerProvider} will be constructed
     * @param sconfig Serialization configuration to use; if null, basic {@link SerializationConfig}
     *  will be constructed
     * @param dconfig Deserialization configuration to use; if null, basic {@link DeserializationConfig}
     *  will be constructed
     */
    public ObjectMapper(JsonFactory jf,
                        SerializerProvider sp, DeserializerProvider dp,
                        SerializationConfig sconfig, DeserializationConfig dconfig)
    {
        /* 02-Mar-2009, tatu: Important: we MUST default to using
         *   the mapping factory, otherwise tree serialization will
         *   have problems with POJONodes.
         * 03-Jan-2010, tatu: and obviously we also must pass 'this',
         *    to create actual linking.
         */
        _jsonFactory = (jf == null) ? new MappingJsonFactory(this) : jf;
        // visibility checker; usually default
        _visibilityChecker = STD_VISIBILITY_CHECKER;
        _serializationConfig = (sconfig != null) ? sconfig :
            new SerializationConfig(DEFAULT_INTROSPECTOR, DEFAULT_ANNOTATION_INTROSPECTOR,
    _visibilityChecker, null);
        _deserializationConfig = (dconfig != null) ? dconfig :
            new DeserializationConfig(DEFAULT_INTROSPECTOR, DEFAULT_ANNOTATION_INTROSPECTOR,
    _visibilityChecker, null);
        _serializerProvider = (sp == null) ? new StdSerializerProvider() : sp;
        _deserializerProvider = (dp == null) ? new StdDeserializerProvider() : dp;

        // Default serializer factory is stateless, can just assign
        _serializerFactory = BeanSerializerFactory.instance;
    }

    /**
```

```
 * Method that will return version information stored in and read from jar
 * that contains this class.
 *
 * @since 1.6
 */
public Version version() {
   return VersionUtil.versionFor(getClass());
}

/**
 * Method for setting specific {@link SerializerFactory} to use
 * for constructing (bean) serializers.
 */
public ObjectMapper setSerializerFactory(SerializerFactory f) {
   _serializerFactory = f;
   return this;
}

/**
 * Method for setting specific {@link SerializerProvider} to use
 * for handling caching of {@link JsonSerializer} instances.
 */
public ObjectMapper setSerializerProvider(SerializerProvider p) {
   _serializerProvider = p;
   return this;
}

/**
 * @since 1.4
 */
public SerializerProvider getSerializerProvider() {
   return _serializerProvider;
}

/**
 * Method for setting specific {@link DeserializerProvider} to use
 * for handling caching of {@link JsonDeserializer} instances.
 */
public ObjectMapper setDeserializerProvider(DeserializerProvider p) {
   _deserializerProvider = p;
   return this;
}

/**
 * @since 1.4
 */
public DeserializerProvider getDeserializerProvider() {
   return _deserializerProvider;
```

```
}

/**
 * Method for specifying {@link JsonNodeFactory} to use for
 * constructing root level tree nodes (via method
 * {@link #createObjectNode}
 *
 * @since 1.2
 */
public ObjectMapper setNodeFactory(JsonNodeFactory f) {
    _deserializationConfig.setNodeFactory(f);
    return this;
}

/**
 * Method for accessing currently configured visibility checker;
 * object used for determining whether given property element
 * (method, field, constructor) can be auto-detected or not.
 *
 * @since 1.5
 */
public VisibilityChecker<?> getVisibilityChecker() {
 return _visibilityChecker;
}

/**
 * Method for setting currently configured visibility checker;
 * object used for determining whether given property element
 * (method, field, constructor) can be auto-detected or not.
 * This default checker is used if no per-class overrides
 * are defined.
 *
 * @since 1.5
 */
public void setVisibilityChecker(VisibilityChecker<?> vc) {
    _visibilityChecker = vc;
}

/**
 * @since 1.6
 */
public SubtypeResolver getSubtypeResolver() {
    if (_subtypeResolver == null) {
        _subtypeResolver = new StdSubtypeResolver();
    }
    return _subtypeResolver;
}
```

```
/**
 * @since 1.6
 */
public void setSubtypeResolver(SubtypeResolver r) {
    _subtypeResolver = r;
}

/**
 * Method for registering specified class as a subtype, so that
 * typename-based resolution can link supertypes to subtypes
 * (as an alternative to using annotations).
 * Type for given class is determined from appropriate annotation;
 * or if missing, default name (unqualified class name)
 *
 * @since 1.6
 */
public void registerSubtypes(Class<?>... classes) {
    getSubtypeResolver().registerSubtypes(classes);
}

/**
 * Method for registering specified class as a subtype, so that
 * typename-based resolution can link supertypes to subtypes
 * (as an alternative to using annotations).
 * Name may be provided as part of argument, but if not will
 * be based on annotations or use default name (unqualified
 * class name).
 *
 * @since 1.6
 */
public void registerSubtypes(NamedType... types) {
    getSubtypeResolver().registerSubtypes(types);
}

/**
 * Method for registering a module that can extend functionality
 * provided by this mapper; for example, by adding providers for
 * custom serializers and deserializers.
 *
 * @param module Module to register
 *
 * @since 1.7
 */
public void registerModule(Module module)
{
    /* Let's ensure we have access to name and version information,
     * even if we do not have immediate use for either. This way we know
     * that they will be available from beginning
```

```
 */
String name = module.getModuleName();
if (name == null) {
   throw new IllegalArgumentException("Module without defined name");
}
Version version = module.version();
if (version == null) {
   throw new IllegalArgumentException("Module without defined version");
}

final ObjectMapper mapper = this;

// And then call registration
module.setupModule(new Module.SetupContext()
{
   // // // Accessors

   @Override
   public Version getMapperVersion() {
      return version();
   }

   public DeserializationConfig getDeserializationConfig() {
      return mapper.getDeserializationConfig();
   }

   public SerializationConfig getSerializationConfig() {
      return mapper.getSerializationConfig();
   }

   @Override
   public SerializationConfig getSeserializationConfig() {
      return getSerializationConfig();
   }

   // // // Methods for registering handlers

   @Override
   public void addSerializers(Serializers s) {
      mapper._serializerFactory = mapper._serializerFactory.withAdditionalSerializers(s);
   }

   @Override
   public void addBeanSerializerModifier(BeanSerializerModifier modifier) {
      mapper._serializerFactory = mapper._serializerFactory.withSerializerModifier(modifier);
   }

   @Override
```

```
        public void addBeanDeserializerModifier(BeanDeserializerModifier modifier) {
            mapper._deserializerProvider = mapper._deserializerProvider.withDeserializerModifier(modifier);
        }

        @Override
        public void addDeserializers(Deserializers d) {
            mapper._deserializerProvider = mapper._deserializerProvider.withAdditionalDeserializers(d);
        }

        @Override
        public void insertAnnotationIntrospector(AnnotationIntrospector ai) {
            mapper._deserializationConfig.insertAnnotationIntrospector(ai);
            mapper._serializationConfig.insertAnnotationIntrospector(ai);
        }

        @Override
        public void appendAnnotationIntrospector(AnnotationIntrospector ai) {
            mapper._deserializationConfig.appendAnnotationIntrospector(ai);
            mapper._serializationConfig.appendAnnotationIntrospector(ai);
        }

        @Override
        public void setMixInAnnotations(Class<?> target, Class<?> mixinSource) {
            mapper._deserializationConfig.addMixInAnnotations(target, mixinSource);
            mapper._serializationConfig.addMixInAnnotations(target, mixinSource);
        }
    });
}

/*
/**********************************************************
/* Access to configuration settings
/**********************************************************
 */

/**
 * Method that returns
 * the shared default {@link SerializationConfig} object
 * that defines configuration settings for serialization.
 * Returned object is "live" meaning that changes will be used
 * for future serialization operations for this mapper when using
 * mapper's default configuration
 */
public SerializationConfig getSerializationConfig() {
    return _serializationConfig;
}

/**
```

```
     * Method that creates a copy of
     * the shared default {@link SerializationConfig} object
     * that defines configuration settings for serialization.
     * Since it is a copy, any changes made to the configuration
     * object will NOT directly affect serialization done using
     * basic serialization methods that use the shared object (that is,
     * ones that do not take separate {@link SerializationConfig}
     * argument.
     *<p>
     * The use case is that of changing object settings of the configuration
     * (like date format being used, see {@link SerializationConfig#setDateFormat}).
     */
    public SerializationConfig copySerializationConfig() {
        return _serializationConfig.createUnshared(_defaultTyper, _visibilityChecker, _subtypeResolver);
    }

    /**
     * Method for replacing the shared default serialization configuration
     * object.
     */
    public ObjectMapper setSerializationConfig(SerializationConfig cfg) {
        _serializationConfig = cfg;
        return this;
    }

    /**
     * Method for changing state of an on/off serialization feature for
     * this object mapper.
     *<p>
     * This is method is basically a shortcut method for calling
     * {@link SerializationConfig#set} on the shared {@link SerializationConfig}
     * object with given arguments.
     */
    public ObjectMapper configure(SerializationConfig.Feature f, boolean state) {
        _serializationConfig.set(f, state);
        return this;
    }

    /**
     * Method that returns
     * the shared default {@link DeserializationConfig} object
     * that defines configuration settings for deserialization.
     * Returned object is "live" meaning that changes will be used
     * for future deserialization operations for this mapper when using
     * mapper's default configuration
     */
    public DeserializationConfig getDeserializationConfig() {
        return _deserializationConfig;
```

```
}

/**
 * Method that creates a copy of
 * the shared default {@link DeserializationConfig} object
 * that defines configuration settings for deserialization.
 * Since it is a copy, any changes made to the configuration
 * object will NOT directly affect deserialization done using
 * basic deserialization methods that use the shared object (that is,
 * ones that do not take separate {@link DeserializationConfig}
 * argument.
 *<p>
 * The use case is that of changing object settings of the configuration
 * (like deserialization problem handler,
 * see {@link DeserializationConfig#addHandler})
 */
public DeserializationConfig copyDeserializationConfig() {
   return _deserializationConfig.createUnshared(_defaultTyper,
         _visibilityChecker, _subtypeResolver);
}

/**
 * Method for replacing the shared default deserialization configuration
 * object.
 */
public ObjectMapper setDeserializationConfig(DeserializationConfig cfg) {
   _deserializationConfig = cfg;
   return this;
}

/**
 * Method for changing state of an on/off deserialization feature for
 * this object mapper.
 *<p>
 * This is method is basically a shortcut method for calling
 * {@link DeserializationConfig#set} on the shared {@link DeserializationConfig}
 * object with given arguments.
 */
public ObjectMapper configure(DeserializationConfig.Feature f, boolean state) {
   _deserializationConfig.set(f, state);
   return this;
}

/**
 * Method that can be used to get hold of {@link JsonFactory} that this
 * mapper uses if it needs to construct {@link JsonParser}s
 * and/or {@link JsonGenerator}s.
 *
```

```
     * @return {@link JsonFactory} that this mapper uses when it needs to
     *   construct Json parser and generators
     */
    public JsonFactory getJsonFactory() { return _jsonFactory; }

    /**
     * Method for changing state of an on/off {@link JsonParser} feature for
     * {@link JsonFactory} instance this object mapper uses.
     *<p>
     * This is method is basically a shortcut method for calling
     * {@link JsonFactory#setParserFeature} on the shared
     * {@link JsonFactory} this mapper uses (which is accessible
     * using {@link #getJsonFactory}).
     *
     * @since 1.2
     */
    public ObjectMapper configure(JsonParser.Feature f, boolean state) {
       _jsonFactory.configure(f, state);
       return this;
    }

    /**
     * Method for changing state of an on/off {@link JsonGenerator} feature for
     * {@link JsonFactory} instance this object mapper uses.
     *<p>
     * This is method is basically a shortcut method for calling
     * {@link JsonFactory#setGeneratorFeature} on the shared
     * {@link JsonFactory} this mapper uses (which is accessible
     * using {@link #getJsonFactory}).
     *
     * @since 1.2
     */
    public ObjectMapper configure(JsonGenerator.Feature f, boolean state) {
       _jsonFactory.configure(f, state);
       return this;
    }

    /**
     * Method that can be used to get hold of {@link JsonNodeFactory}
     * that this mapper will use when directly constructing
     * root {@link JsonNode} instances for Trees.
     *<p>
     * Note: this is just a shortcut for calling
     *<pre>
     *   getDeserializationConfig().getNodeFactory()
     *</pre>
     *
     * @since 1.2
```

```
 */
public JsonNodeFactory getNodeFactory() {
   return _deserializationConfig.getNodeFactory();
}

/*
/**********************************************************
/* Type information configuration (1.5+)
/**********************************************************
 */

/**
 * Convenience method that is equivalent to calling
 *<pre>
 *  enableObjectTyping(DefaultTyping.OBJECT_AND_NON_CONCRETE);
 *</pre>
 */
public ObjectMapper enableDefaultTyping() {
   return enableDefaultTyping(DefaultTyping.OBJECT_AND_NON_CONCRETE);
}

/**
 * Convenience method that is equivalent to calling
 *<pre>
 *  enableObjectTyping(dti, JsonTypeInfo.As.WRAPPER_ARRAY);
 *</pre>
 */
public ObjectMapper enableDefaultTyping(DefaultTyping dti) {
   return enableDefaultTyping(dti, JsonTypeInfo.As.WRAPPER_ARRAY);
}

/**
 * Method for enabling automatic inclusion of type information, needed
 * for proper deserialization of polymorphic types (unless types
 * have been annotated with {@link org.codehaus.jackson.annotate.JsonTypeInfo}).
 *
 * @param applicability Defines kinds of types for which additional type information
 *    is added; see {@link DefaultTyping} for more information.
 */
public ObjectMapper enableDefaultTyping(DefaultTyping applicability, JsonTypeInfo.As includeAs)
{
   TypeResolverBuilder<?> typer = new DefaultTypeResolverBuilder(applicability);
   // we'll always use full class name, when using defaulting
   typer = typer.init(JsonTypeInfo.Id.CLASS, null);
   typer = typer.inclusion(includeAs);
   return setDefaultTyping(typer);
}
```

```
/**
 * Method for enabling automatic inclusion of type information -- needed
 * for proper deserialization of polymorphic types (unless types
 * have been annotated with {@link org.codehaus.jackson.annotate.JsonTypeInfo}) --
 * using "As.PROPERTY" inclusion mechanism and specified property name
 * to use for inclusion (default being "@class" since default type information
 * always uses class name as type identifier)
 *
 * @since 1.7
 */
public ObjectMapper enableDefaultTypingAsProperty(DefaultTyping applicability, String propertyName)
{
   TypeResolverBuilder<?> typer = new DefaultTypeResolverBuilder(applicability);
   // we'll always use full class name, when using defaulting
   typer = typer.init(JsonTypeInfo.Id.CLASS, null);
   typer = typer.inclusion(JsonTypeInfo.As.PROPERTY);
   typer = typer.typeProperty(propertyName);
   return setDefaultTyping(typer);
}

/**
 * Method for disabling automatic inclusion of type information; if so, only
 * explicitly annotated types (ones with
 * {@link org.codehaus.jackson.annotate.JsonTypeInfo}) will have
 * additional embedded type information.
 */
public ObjectMapper disableDefaultTyping() {
   return setDefaultTyping(null);
}

/**
 * @deprecated Typo in name -- use {@link #setDefaultTyping} instead.
 */
@Deprecated
public ObjectMapper setDefaltTyping(TypeResolverBuilder<?> typer) {
   _defaultTyper = typer;
   return this;
}

/**
 * Method for enabling automatic inclusion of type information, using
 * specified handler object for determining which types this affects,
 * as well as details of how information is embedded.
 *
 * @param typer Type information inclusion handler
 *
 *
 */
```

```
  public ObjectMapper setDefaultTyping(TypeResolverBuilder<?> typer) {
     _defaultTyper = typer;
     return this;
  }

  /*
  /**********************************************************
  /* Public API (from ObjectCodec): deserialization
  /* (mapping from JSON to Java types);
  /* main methods
  /**********************************************************
   */

  /**
   * Method to deserialize JSON content into a non-container
   * type (it can be an array type, however): typically a bean, array
   * or a wrapper type (like {@link java.lang.Boolean}).
   *<p>
   * Note: this method should NOT be used if the result type is a
   * container ({@link java.util.Collection} or {@link java.util.Map}.
   * The reason is that due to type erasure, key and value types
   * can not be introspected when using this method.
   */
  @Override
  @SuppressWarnings("unchecked")
  public <T> T readValue(JsonParser jp, Class<T> valueType)
     throws IOException, JsonParseException, JsonMappingException
  {
// !!! TODO
//     _setupClassLoaderForDeserialization(valueType);
     return (T) _readValue(copyDeserializationConfig(), jp, TypeFactory.type(valueType));
  }

  /**
   * Method to deserialize Json content into a non-container
   * type (it can be an array type, however): typically a bean, array
   * or a wrapper type (like {@link java.lang.Boolean}).
   *<p>
   * Note: this method should NOT be used if the result type is a
   * container ({@link java.util.Collection} or {@link java.util.Map}.
   * The reason is that due to type erasure, key and value types
   * can not be introspected when using this method.
   * @since 1.1
   *
   * @param cfg Specific deserialization configuration to use for
   *   this operation. Note that not all config settings can
   *   be changed on per-operation basis: some changeds only take effect
   *   before calling the operation for the first time (for the mapper
```

```
     *   instance)
     */
    @SuppressWarnings("unchecked")
    public <T> T readValue(JsonParser jp, Class<T> valueType,
                           DeserializationConfig cfg)
        throws IOException, JsonParseException, JsonMappingException
    {
     // !!! TODO
//    _setupClassLoaderForDeserialization(valueType);
        return (T) _readValue(cfg, jp, TypeFactory.type(valueType));
    }

    /**
     * Method to deserialize Json content into a Java type, reference
     * to which is passed as argument. Type is passed using so-called
     * "super type token" (see )
     * and specifically needs to be used if the root type is a
     * parameterized (generic) container type.
     */
    @Override
    @SuppressWarnings("unchecked")
    public <T> T readValue(JsonParser jp, TypeReference<?> valueTypeRef)
        throws IOException, JsonParseException, JsonMappingException
    {
        return (T) _readValue(copyDeserializationConfig(), jp, TypeFactory.type(valueTypeRef));
    }

    /**
     * Method to deserialize Json content into a Java type, reference
     * to which is passed as argument. Type is passed using so-called
     * "super type token" (see )
     * and specifically needs to be used if the root type is a
     * parameterized (generic) container type.
     *
     * @param cfg Specific deserialization configuration to use for
     *   this operation. Note that not all config settings can
     *   be changed on per-operation basis: some changeds only take effect
     *   before calling the operation for the first time (for the mapper
     *   instance)
     *
     * @since 1.1
     */
    @SuppressWarnings("unchecked")
    public <T> T readValue(JsonParser jp, TypeReference<?> valueTypeRef,
                           DeserializationConfig cfg)
        throws IOException, JsonParseException, JsonMappingException
    {
        return (T) _readValue(cfg, jp, TypeFactory.type(valueTypeRef));
```

```
}

/**
 * Method to deserialize Json content into a Java type, reference
 * to which is passed as argument. Type is passed using
 * Jackson specific type; instance of which can be constructed using
 * {@link TypeFactory}.
 */
@Override
@SuppressWarnings("unchecked")
public <T> T readValue(JsonParser jp, JavaType valueType)
   throws IOException, JsonParseException, JsonMappingException
{
   return (T) _readValue(copyDeserializationConfig(), jp, valueType);
}

/**
 * Method to deserialize Json content into a Java type, reference
 * to which is passed as argument. Type is passed using
 * Jackson specific type; instance of which can be constructed using
 * {@link TypeFactory}.
 *
 * @param cfg Specific deserialization configuration to use for
 *   this operation. Note that not all config settings can
 *   be changed on per-operation basis: some changeds only take effect
 *   before calling the operation for the first time (for the mapper
 *   instance)
 *
 * @since 1.1
 */
@SuppressWarnings("unchecked")
public <T> T readValue(JsonParser jp, JavaType valueType,
                       DeserializationConfig cfg)
   throws IOException, JsonParseException, JsonMappingException
{
   return (T) _readValue(cfg, jp, valueType);
}

/**
 * Method to deserialize JSON content as tree expressed
 * using set of {@link JsonNode} instances. Returns
 * root of the resulting tree (where root can consist
 * of just a single node if the current event is a
 * value event, not container).
 */
@Override
public JsonNode readTree(JsonParser jp)
   throws IOException, JsonProcessingException
```

```
{
   return readTree(jp, copyDeserializationConfig());
}

/**
 * Method to deserialize JSON content as tree expressed
 * using set of {@link JsonNode} instances. Returns
 * root of the resulting tree (where root can consist
 * of just a single node if the current event is a
 * value event, not container).
 *
 * @param cfg Specific deserialization configuration to use for
 *   this operation. Note that not all config settings can
 *   be changed on per-operation basis: some changeds only take effect
 *   before calling the operation for the first time (for the mapper
 *   instance)
 *
 * @since 1.1
 */
public JsonNode readTree(JsonParser jp, DeserializationConfig cfg)
   throws IOException, JsonProcessingException
{
   /* 02-Mar-2009, tatu: One twist; deserialization provider
    *   will map Json null straight into Java null. But what
    *   we want to return is the "null node" instead.
    */
   JsonNode n = (JsonNode) _readValue(cfg, jp, JSON_NODE_TYPE);
   return (n == null) ? NullNode.instance : n;
}

/**
 * Method to deserialize JSON content as tree expressed
 * using set of {@link JsonNode} instances.
 * Returns root of the resulting tree (where root can consist
 * of just a single node if the current event is a
 * value event, not container).
 *
 * @param in Input stream used to read JSON content
 *   for building the JSON tree.
 *
 * @since 1.3
 */
public JsonNode readTree(InputStream in)
   throws IOException, JsonProcessingException
{
   JsonNode n = (JsonNode) readValue(in, JSON_NODE_TYPE);
   return (n == null) ? NullNode.instance : n;
}
```

```
/**
 * Method to deserialize JSON content as tree expressed
 * using set of {@link JsonNode} instances.
 * Returns root of the resulting tree (where root can consist
 * of just a single node if the current event is a
 * value event, not container).
 *
 * @param r Reader used to read JSON content
 *   for building the JSON tree.
 *
 * @since 1.3
 */
public JsonNode readTree(Reader r)
   throws IOException, JsonProcessingException
{
   JsonNode n = (JsonNode) readValue(r, JSON_NODE_TYPE);
   return (n == null) ? NullNode.instance : n;
}

/**
 * Method to deserialize JSON content as tree expressed
 * using set of {@link JsonNode} instances.
 * Returns root of the resulting tree (where root can consist
 * of just a single node if the current event is a
 * value event, not container).
 *
 * @param content JSON content to parse
 *   for building the JSON tree.
 *
 * @since 1.3
 */
public JsonNode readTree(String content)
   throws IOException, JsonProcessingException
{
   JsonNode n = (JsonNode) readValue(content, JSON_NODE_TYPE);
   return (n == null) ? NullNode.instance : n;
}

/*
/**********************************************************
/* Public API (from ObjectCodec): serialization
/* (mapping from Java types to Json)
/**********************************************************
 */

/**
 * Method that can be used to serialize any Java value as
```

```
     * JSON output, using provided {@link JsonGenerator}.
     */
    @Override
    public void writeValue(JsonGenerator jgen, Object value)
       throws IOException, JsonGenerationException, JsonMappingException
    {
       SerializationConfig config = copySerializationConfig();
       if (config.isEnabled(SerializationConfig.Feature.CLOSE_CLOSEABLE) && (value instanceof Closeable)) {
          _writeCloseableValue(jgen, value, config);
       } else {
          _serializerProvider.serializeValue(config, jgen, value, _serializerFactory);
          if (config.isEnabled(SerializationConfig.Feature.FLUSH_AFTER_WRITE_VALUE)) {
             jgen.flush();
          }
       }
    }

    /**
     * Method that can be used to serialize any Java value as
     * JSON output, using provided {@link JsonGenerator},
     * configured as per passed configuration object.
     *
     * @since 1.1
     */
    public void writeValue(JsonGenerator jgen, Object value, SerializationConfig config)
       throws IOException, JsonGenerationException, JsonMappingException
    {
       // [JACKSON-282] Consider java.util.Closeable
       if (config.isEnabled(SerializationConfig.Feature.CLOSE_CLOSEABLE) && (value instanceof Closeable)) {
          _writeCloseableValue(jgen, value, config);
       } else {
          _serializerProvider.serializeValue(config, jgen, value, _serializerFactory);
          if (config.isEnabled(SerializationConfig.Feature.FLUSH_AFTER_WRITE_VALUE)) {
             jgen.flush();
          }
       }
    }

    /**
     * Method to serialize given JSON Tree, using generator
     * provided.
     */
    @Override
    public void writeTree(JsonGenerator jgen, JsonNode rootNode)
       throws IOException, JsonProcessingException
    {
       SerializationConfig config = copySerializationConfig();
       _serializerProvider.serializeValue(config, jgen, rootNode, _serializerFactory);
```

```
    if (config.isEnabled(SerializationConfig.Feature.FLUSH_AFTER_WRITE_VALUE)) {
        jgen.flush();
    }
}

/**
 * Method to serialize given Json Tree, using generator
 * provided.
 *
 * @since 1.1
 */
public void writeTree(JsonGenerator jgen, JsonNode rootNode,
                      SerializationConfig cfg)
    throws IOException, JsonProcessingException
{
    _serializerProvider.serializeValue(cfg, jgen, rootNode, _serializerFactory);
    if (cfg.isEnabled(SerializationConfig.Feature.FLUSH_AFTER_WRITE_VALUE)) {
        jgen.flush();
    }
}

/*
/**********************************************************
/* Public API (from ObjectCodec): Tree Model support
/**********************************************************
 */

/**
 *<p>
 * Note: return type is co-variant, as basic ObjectCodec
 * abstraction can not refer to concrete node types (as it's
 * part of core package, whereas impls are part of mapper
 * package)
 *
 * @since 1.2
 */
@Override
public ObjectNode createObjectNode() {
    return _deserializationConfig.getNodeFactory().objectNode();
}

/**
 *<p>
 * Note: return type is co-variant, as basic ObjectCodec
 * abstraction can not refer to concrete node types (as it's
 * part of core package, whereas impls are part of mapper
 * package)
 *
```

```
 * @since 1.2
 */
@Override
public ArrayNode createArrayNode() {
   return _deserializationConfig.getNodeFactory().arrayNode();
}

/**
 * Method for constructing a {@link JsonParser} out of JSON tree
 * representation.
 *
 * @param n Root node of the tree that resulting parser will read from
 *
 * @since 1.3
 */
@Override
public JsonParser treeAsTokens(JsonNode n)
{
   return new TreeTraversingParser(n, this);
}

/**
 * Convenience conversion method that will bind data given JSON tree
 * contains into specific value (usually bean) type.
 *<p>
 * Equivalent to:
 *<pre>
 *   objectMapper.convertValue(n, valueClass);
 *</pre>
 */
@Override
public <T> T treeToValue(JsonNode n, Class<T> valueType)
   throws IOException, JsonParseException, JsonMappingException
{
   JsonParser jp = treeAsTokens(n);
   return readValue(jp, valueType);
}

/**
 * Reverse of {@link #treeToValue}; given a value (usually bean), will
 * construct equivalent JSON Tree representation. Functionally same
 * as if serializing value into JSON and parsing JSON as tree, but
 * more efficient.
 *
 * @param <T> Actual node type; usually either basic {@link JsonNode} or
 *  {@link org.codehaus.jackson.node.ObjectNode}
 * @param fromValue Bean value to convert
 * @return Root node of the resulting JSON tree
```

```
     *
     * @since 1.6
     */
    @SuppressWarnings("unchecked")
    public <T extends JsonNode> T valueToTree(Object fromValue)
        throws IllegalArgumentException
    {
        if (fromValue == null) return null;
        TokenBuffer buf = new TokenBuffer(this);
        JsonNode result;
        try {
            writeValue(buf, fromValue);
            JsonParser jp = buf.asParser();
            result = readTree(jp);
            jp.close();
        } catch (IOException e) { // should not occur, no real i/o...
            throw new IllegalArgumentException(e.getMessage(), e);
        }
        return (T) result;
    }

    /*
    /**********************************************************
    /* Extended Public API, accessors
    /**********************************************************
     */

    /**
     * Method that can be called to check whether mapper thinks
     * it could serialize an instance of given Class.
     * Check is done
     * by checking whether a serializer can be found for the type.
     *
     * @return True if mapper can find a serializer for instances of
     *  given class (potentially serializable), false otherwise (not
     *  serializable)
     */
    public boolean canSerialize(Class<?> type)
    {
        return _serializerProvider.hasSerializerFor(_serializationConfig, type, _serializerFactory);
    }

    /**
     * Method that can be called to check whether mapper thinks
     * it could deserialize an Object of given type.
     * Check is done
     * by checking whether a deserializer can be found for the type.
     *
```

```
     * @return True if mapper can find a serializer for instances of
     *  given class (potentially serializable), false otherwise (not
     *  serializable)
     */
    public boolean canDeserialize(JavaType type)
    {
        return _deserializerProvider.hasValueDeserializerFor(_deserializationConfig, type);
    }

    /*
    /**********************************************************
    /* Extended Public API, deserialization,
    /* convenience methods
    /**********************************************************
     */

    @SuppressWarnings("unchecked")
    public <T> T readValue(File src, Class<T> valueType)
        throws IOException, JsonParseException, JsonMappingException
    {
     // !!! TODO
//     _setupClassLoaderForDeserialization(valueType);
        return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), TypeFactory.type(valueType));
    }

    @SuppressWarnings("unchecked")
    public <T> T readValue(File src, TypeReference valueTypeRef)
        throws IOException, JsonParseException, JsonMappingException
    {
        return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), TypeFactory.type(valueTypeRef));
    }

    @SuppressWarnings("unchecked")
    public <T> T readValue(File src, JavaType valueType)
        throws IOException, JsonParseException, JsonMappingException
    {
        return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), valueType);
    }

    @SuppressWarnings("unchecked")
    public <T> T readValue(URL src, Class<T> valueType)
        throws IOException, JsonParseException, JsonMappingException
    {
     // !!! TODO
//     _setupClassLoaderForDeserialization(valueType);
        return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), TypeFactory.type(valueType));
    }
```

```
    @SuppressWarnings("unchecked")
    public <T> T readValue(URL src, TypeReference valueTypeRef)
        throws IOException, JsonParseException, JsonMappingException
    {
        return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), TypeFactory.type(valueTypeRef));
    }

    @SuppressWarnings("unchecked")
    public <T> T readValue(URL src, JavaType valueType)
        throws IOException, JsonParseException, JsonMappingException
    {
        return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), valueType);
    }

    @SuppressWarnings("unchecked")
    public <T> T readValue(String content, Class<T> valueType)
        throws IOException, JsonParseException, JsonMappingException
    {
     // !!! TODO
//      _setupClassLoaderForDeserialization(valueType);
        return (T) _readMapAndClose(_jsonFactory.createJsonParser(content), TypeFactory.type(valueType));
    }

    @SuppressWarnings("unchecked")
    public <T> T readValue(String content, TypeReference valueTypeRef)
        throws IOException, JsonParseException, JsonMappingException
    {
        return (T) _readMapAndClose(_jsonFactory.createJsonParser(content), TypeFactory.type(valueTypeRef));
    }

    @SuppressWarnings("unchecked")
    public <T> T readValue(String content, JavaType valueType)
        throws IOException, JsonParseException, JsonMappingException
    {
        return (T) _readMapAndClose(_jsonFactory.createJsonParser(content), valueType);
    }

    @SuppressWarnings("unchecked")
    public <T> T readValue(Reader src, Class<T> valueType)
        throws IOException, JsonParseException, JsonMappingException
    {
     // !!! TODO
//      _setupClassLoaderForDeserialization(valueType);
        return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), TypeFactory.type(valueType));
    }

    @SuppressWarnings("unchecked")
    public <T> T readValue(Reader src, TypeReference valueTypeRef)
```

```
    throws IOException, JsonParseException, JsonMappingException
  {
    return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), TypeFactory.type(valueTypeRef));
  }

  @SuppressWarnings("unchecked")
  public <T> T readValue(Reader src, JavaType valueType)
    throws IOException, JsonParseException, JsonMappingException
  {
    return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), valueType);
  }

  @SuppressWarnings("unchecked")
  public <T> T readValue(InputStream src, Class<T> valueType)
    throws IOException, JsonParseException, JsonMappingException
  {
   // !!! TODO
//    _setupClassLoaderForDeserialization(valueType);
    return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), TypeFactory.type(valueType));
  }

  @SuppressWarnings("unchecked")
  public <T> T readValue(InputStream src, TypeReference valueTypeRef)
    throws IOException, JsonParseException, JsonMappingException
  {
    return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), TypeFactory.type(valueTypeRef));
  }

  @SuppressWarnings("unchecked")
  public <T> T readValue(InputStream src, JavaType valueType)
    throws IOException, JsonParseException, JsonMappingException
  {
    return (T) _readMapAndClose(_jsonFactory.createJsonParser(src), valueType);
  }

  @SuppressWarnings("unchecked")
  public <T> T readValue(byte[] src, int offset, int len,
                         Class<T> valueType)
    throws IOException, JsonParseException, JsonMappingException
  {
   // !!! TODO
//    _setupClassLoaderForDeserialization(valueType);
    return (T) _readMapAndClose(_jsonFactory.createJsonParser(src, offset, len), TypeFactory.type(valueType));
  }

  @SuppressWarnings("unchecked")
  public <T> T readValue(byte[] src, int offset, int len,
                         TypeReference valueTypeRef)
```

```
    throws IOException, JsonParseException, JsonMappingException
  {
    return (T) _readMapAndClose(_jsonFactory.createJsonParser(src, offset, len),
TypeFactory.type(valueTypeRef));
  }

  @SuppressWarnings("unchecked")
  public <T> T readValue(byte[] src, int offset, int len,
                         JavaType valueType)
    throws IOException, JsonParseException, JsonMappingException
  {
    return (T) _readMapAndClose(_jsonFactory.createJsonParser(src, offset, len), valueType);
  }

  /**
   * Convenience method for converting results from given JSON tree into given
   * value type. Basically short-cut for:
   *<pre>
   *   mapper.readValue(root.traverse(), valueType);
   *</pre>
   *
   * @since 1.6
   */
  @SuppressWarnings("unchecked")
  public <T> T readValue(JsonNode root, Class<T> valueType)
    throws IOException, JsonParseException, JsonMappingException
  {
   // !!! TODO
//    _setupClassLoaderForDeserialization(valueType);
    return (T) _readValue(copyDeserializationConfig(), root.traverse(), TypeFactory.type(valueType));
  }

  /**
   * Convenience method for converting results from given JSON tree into given
   * value type. Basically short-cut for:
   *<pre>
   *   mapper.readValue(root.traverse(), valueType);
   *</pre>
   *
   * @since 1.6
   */
  @SuppressWarnings("unchecked")
  public <T> T readValue(JsonNode root, TypeReference valueTypeRef)
    throws IOException, JsonParseException, JsonMappingException
  {
    return (T) _readValue(copyDeserializationConfig(), root.traverse(), TypeFactory.type(valueTypeRef));
  }
```

```
/**
 * Convenience method for converting results from given JSON tree into given
 * value type. Basically short-cut for:
 *<pre>
 *   mapper.readValue(root.traverse(), valueType);
 *</pre>
 *
 * @since 1.6
 */
@SuppressWarnings("unchecked")
public <T> T readValue(JsonNode root, JavaType valueType)
   throws IOException, JsonParseException, JsonMappingException
{
   return (T) _readValue(copyDeserializationConfig(), root.traverse(), valueType);
}

/*
/**********************************************************
/* Extended Public API: serialization
/* (mapping from Java types to Json)
/**********************************************************
 */

/**
 * Method that can be used to serialize any Java value as
 * JSON output, written to File provided.
 */
public void writeValue(File resultFile, Object value)
   throws IOException, JsonGenerationException, JsonMappingException
{
   _configAndWriteValue(_jsonFactory.createJsonGenerator(resultFile, JsonEncoding.UTF8), value);
}

/**
 * Method that can be used to serialize any Java value as
 * JSON output, using output stream provided (using encoding
 * {@link JsonEncoding#UTF8}).
 *<p>
 * Note: method does not close the underlying stream explicitly
 * here; however, {@link JsonFactory} this mapper uses may choose
 * to close the stream depending on its settings (by default,
 * it will try to close it when {@link JsonGenerator} we construct
 * is closed).
 */
public void writeValue(OutputStream out, Object value)
   throws IOException, JsonGenerationException, JsonMappingException
{
   _configAndWriteValue(_jsonFactory.createJsonGenerator(out, JsonEncoding.UTF8), value);
```

```
}

/**
 * Method that can be used to serialize any Java value as
 * JSON output, using Writer provided.
 *<p>
 * Note: method does not close the underlying stream explicitly
 * here; however, {@link JsonFactory} this mapper uses may choose
 * to close the stream depending on its settings (by default,
 * it will try to close it when {@link JsonGenerator} we construct
 * is closed).
 */
public void writeValue(Writer w, Object value)
   throws IOException, JsonGenerationException, JsonMappingException
{
   _configAndWriteValue(_jsonFactory.createJsonGenerator(w), value);
}

/**
 * Method that can be used to serialize any Java value as
 * a String. Functionally equivalent to calling
 * {@link #writeValue(Writer,Object)} with {@link java.io.StringWriter}
 * and constructing String, but more efficient.
 *
 * @since 1.3
 */
public String writeValueAsString(Object value)
   throws IOException, JsonGenerationException, JsonMappingException
{
   // alas, we have to pull the recycler directly here...
   SegmentedStringWriter sw = new SegmentedStringWriter(_jsonFactory._getBufferRecycler());
   _configAndWriteValue(_jsonFactory.createJsonGenerator(sw), value);
   return sw.getAndClear();
}

/**
 * Method that can be used to serialize any Java value as
 * a byte array. Functionally equivalent to calling
 * {@link #writeValue(Writer,Object)} with {@link java.io.ByteArrayOutputStream}
 * and getting bytes, but more efficient.
 * Encoding used will be UTF-8.
 *
 * @since 1.5
 */
public byte[] writeValueAsBytes(Object value)
   throws IOException, JsonGenerationException, JsonMappingException
{
   ByteArrayBuilder bb = new ByteArrayBuilder(_jsonFactory._getBufferRecycler());
```

```
    _configAndWriteValue(_jsonFactory.createJsonGenerator(bb, JsonEncoding.UTF8), value);
    byte[] result = bb.toByteArray();
    bb.release();
    return result;
}

/*
/**********************************************************
/* Extended Public API: serialization using JSON Views
/* (since version 1.4)
/*
/* NOTE: as of version 1.5, should use ObjectWriter
/* instead
/**********************************************************
 */

/**
 * Method for serializing given object using specified view.
 * Note that this method is essentially just a shortcut: view to use is
 * by {@link SerializationConfig} and so this method just assigns
 * given view to a copy of default configuration of this
 * mapper. So to use other kinds of destinations, just call
 * {@link #copySerializationConfig}, call
 * {@link SerializationConfig#setSerializationView} method on it,
 * and pass that configuration to other methods.
 *
 * @param jgen Generator to use for writing JSON content
 * @param value Value to serialize
 * @param viewClass (optional) Identifier for View to use. If null,
 *   equivalent to passing <code>Object.class</code>; both of which
 *   mean "default view" (all properties always included)
 *
 * @deprecated Use {@link #viewWriter} instead
 */
@Deprecated
public void writeValueUsingView(JsonGenerator jgen, Object value, Class<?> viewClass)
    throws IOException, JsonGenerationException, JsonMappingException
{
    _configAndWriteValue(jgen, value, viewClass);
}

/**
 * Method for serializing given object using specified view.
 * As with {@link #writeValueUsingView(JsonGenerator,Object,Class)},
 * this is a short-cut method.
 *
 * @param w Writer used for writing JSON content
 * @param value Value to serialize
```

```
     * @param viewClass (optional) Identifier for View to use. If null,
     *   equivalent to passing <code>Object.class</code>; both of which
     *   mean "default view" (all properties always included)
     *
     * @deprecated Use {@link #viewWriter} instead
     */
    @Deprecated
    public void writeValueUsingView(Writer w, Object value, Class<?> viewClass)
       throws IOException, JsonGenerationException, JsonMappingException
    {
       _configAndWriteValue(_jsonFactory.createJsonGenerator(w), value, viewClass);
    }

    /**
     * Method for serializing given object using specified view.
     * As with {@link #writeValueUsingView(JsonGenerator,Object,Class)},
     * this is a short-cut method.
     *
     * @param out Output stream used for writing JSON content
     * @param value Value to serialize
     * @param viewClass (optional) Identifier for View to use. If null,
     *   equivalent to passing <code>Object.class</code>; both of which
     *   mean "default view" (all properties always included)
     *
     * @deprecated Use {@link #viewWriter} instead
     */
    @Deprecated
    public void writeValueUsingView(OutputStream out, Object value, Class<?> viewClass)
       throws IOException, JsonGenerationException, JsonMappingException
    {
       _configAndWriteValue(_jsonFactory.createJsonGenerator(out, JsonEncoding.UTF8), value, viewClass);
    }

    /*
    /**********************************************************
    /* Extended Public API: constructing ObjectWriters
    /* for more advanced configuration
    /**********************************************************
     */

    /**
     * Convenience method for constructing {@link ObjectWriter}
     * with default settings.
     *
     * @since 1.6
     */
    public ObjectWriter writer() {
       return new ObjectWriter(this, /*view*/null, /*type*/ null, /*PrettyPrinter*/null);
```

```
}

/**
 * Factory method for constructing {@link ObjectWriter} that will
 * serialize objects using specified JSON View (filter).
 *
 * @since 1.5
 */
public ObjectWriter viewWriter(Class<?> serializationView) {
    return new ObjectWriter(this, serializationView,
            /*type*/ null, /*PrettyPrinter*/null);
}

/**
 * Factory method for constructing {@link ObjectWriter} that will
 * serialize objects using specified root type, instead of actual
 * runtime type of value. Type must be a super-type of runtime
 * type.
 *
 * @since 1.5
 */
public ObjectWriter typedWriter(Class<?> rootType) {
    JavaType t = (rootType == null) ? null : TypeFactory.type(rootType);
    return new ObjectWriter(this, null, t, /*PrettyPrinter*/null);
}

/**
 * Factory method for constructing {@link ObjectWriter} that will
 * serialize objects using specified root type, instead of actual
 * runtime type of value. Type must be a super-type of runtime type.
 *
 * @since 1.5
 */
public ObjectWriter typedWriter(JavaType rootType) {
    return new ObjectWriter(this, null, rootType, /*PrettyPrinter*/null);
}

/**
 * Factory method for constructing {@link ObjectWriter} that will
 * serialize objects using specified root type, instead of actual
 * runtime type of value. Type must be a super-type of runtime type.
 *
 * @since 1.7
 */
public ObjectWriter typedWriter(TypeReference<?> rootType) {
    JavaType t = (rootType == null) ? null : TypeFactory.type(rootType);
    return new ObjectWriter(this, null, t, /*PrettyPrinter*/null);
}
```

```
/**
 * Factory method for constructing {@link ObjectWriter} that will
 * serialize objects using specified pretty printer for indentation
 * (or if null, no pretty printer)
 *
 * @since 1.5
 */
public ObjectWriter prettyPrintingWriter(PrettyPrinter pp) {
    if (pp == null) { // need to use a marker to indicate explicit disabling of pp
        pp = ObjectWriter.NULL_PRETTY_PRINTER;
    }
    return new ObjectWriter(this, null, /*root type*/ null, pp);
}

/**
 * Factory method for constructing {@link ObjectWriter} that will
 * serialize objects using the default pretty printer for indentation
 *
 * @since 1.5
 */
public ObjectWriter defaultPrettyPrintingWriter() {
    return new ObjectWriter(this, null, /*root type*/ null, _defaultPrettyPrinter());
}

public ObjectWriter filteredWriter(FilterProvider filterProvider) {
    return new ObjectWriter(this, filterProvider);
}

/*
/**********************************************************
/* Extended Public API: constructing ObjectReaders
/* for more advanced configuration
/**********************************************************
 */

/**
 * Factory method for constructing {@link ObjectReader} with
 * default settings.
 *
 * @since 1.6
 */
public ObjectReader reader() {
    return new ObjectReader(this, null, null);
}

/**
 * Factory method for constructing {@link ObjectReader} that will
```

```
     * update given Object (usually Bean, but can be a Collection or Map
     * as well, but NOT an array) with JSON data. Deserialization occurs
     * normally except that the root-level value in JSON is not used for
     * instantiating a new object; instead give updateable object is used
     * as root.
     * Runtime type of value object is used for locating deserializer,
     * unless overridden by other factory methods of {@link ObjectReader}
     *
     * @since 1.6
     */
    public ObjectReader updatingReader(Object valueToUpdate)
    {
        JavaType t = TypeFactory.type(valueToUpdate.getClass());
        return new ObjectReader(this, t, valueToUpdate);
    }

    /**
     * Factory method for constructing {@link ObjectReader} that will
     * read or update instances of specified type
     *
     * @since 1.6
     */
    public ObjectReader reader(JavaType type)
    {
        return new ObjectReader(this, type, null);
    }

    /**
     * Factory method for constructing {@link ObjectReader} that will
     * read or update instances of specified type
     *
     * @since 1.6
     */
    public ObjectReader reader(Class<?> type)
    {
        return reader(TypeFactory.type(type));
    }

    /**
     * Factory method for constructing {@link ObjectReader} that will
     * read or update instances of specified type
     *
     * @since 1.6
     */
    public ObjectReader reader(TypeReference<?> type)
    {
        return reader(TypeFactory.type(type));
    }
```

```
/**
 * Factory method for constructing {@link ObjectReader} that will
 * use specified {@link JsonNodeFactory} for constructing JSON trees.
 *
 * @since 1.6
 */
public ObjectReader reader(JsonNodeFactory f)
{
   return new ObjectReader(this, null, null).withNodeFactory(f);
}

/*
/**********************************************************
/* Extended Public API: convenience type conversion
/**********************************************************
 */

/**
 * Convenience method for doing two-step conversion from given value, into
 * instance of given value type. This is functionality equivalent to first
 * serializing given value into JSON, then binding JSON data into value
 * of given type, but may be executed without fully serializing into
 * JSON. Same converters (serializers, deserializers) will be used as for
 * data binding, meaning same object mapper configuration works.
 *
 * @throws IllegalArgumentException If conversion fails due to incompatible type;
 *    if so, root cause will contain underlying checked exception data binding
 *    functionality threw
 */
@SuppressWarnings("unchecked")
public <T> T convertValue(Object fromValue, Class<T> toValueType)
   throws IllegalArgumentException
{
   return (T) _convert(fromValue, TypeFactory.type(toValueType));
}

@SuppressWarnings("unchecked")
public <T> T convertValue(Object fromValue, TypeReference toValueTypeRef)
   throws IllegalArgumentException
{
   return (T) _convert(fromValue, TypeFactory.type(toValueTypeRef));
}

@SuppressWarnings("unchecked")
public <T> T convertValue(Object fromValue, JavaType toValueType)
   throws IllegalArgumentException
{
```

```
    return (T) _convert(fromValue, toValueType);
}

protected Object _convert(Object fromValue, JavaType toValueType)
    throws IllegalArgumentException
{
    // sanity check for null first:
    if (fromValue == null) return null;
    /* Then use TokenBuffer, which is a JsonGenerator:
     * (see [JACKSON-175])
     */
    TokenBuffer buf = new TokenBuffer(this);
    try {
        writeValue(buf, fromValue);
        // and provide as with a JsonParser for contents as well!
        JsonParser jp = buf.asParser();
        Object result = readValue(jp, toValueType);
        jp.close();
        return result;
    } catch (IOException e) { // should not occur, no real i/o...
        throw new IllegalArgumentException(e.getMessage(), e);
    }
}

/*
/**********************************************************
/* Extended Public API: JSON Schema generation
/**********************************************************
 */

/**
 * Generate <a href="http://json-schema.org/">Json-schema</a>
 * instance for specified class.
 *
 * @param t The class to generate schema for
 * @return Constructed JSON schema.
 */
public JsonSchema generateJsonSchema(Class<?> t)
        throws JsonMappingException
{
    return generateJsonSchema(t, copySerializationConfig());
}

/**
 * Generate <a href="http://json-schema.org/">Json-schema</a>
 * instance for specified class, using specific
 * serialization configuration
 *
```

```
     * @param t The class to generate schema for
     * @return Constructed JSON schema.
     */
    public JsonSchema generateJsonSchema(Class<?> t, SerializationConfig cfg)
            throws JsonMappingException
    {
        return _serializerProvider.generateJsonSchema(t, cfg, _serializerFactory);
    }

    /*
    /**********************************************************
    /* Internal methods, overridable
    /**********************************************************
     */

    /**
     * Helper method that should return default pretty-printer to
     * use for generators constructed by this mapper, when instructed
     * to use default pretty printer.
     *
     * @since 1.7
     */
    protected PrettyPrinter _defaultPrettyPrinter() {
        return new DefaultPrettyPrinter();
    }

    /**
     * Method called to configure the generator as necessary and then
     * call write functionality
     */
    protected final void _configAndWriteValue(JsonGenerator jgen, Object value)
        throws IOException, JsonGenerationException, JsonMappingException
    {
        SerializationConfig cfg = copySerializationConfig();
        // [JACKSON-96]: allow enabling pretty printing for ObjectMapper directly
        if (cfg.isEnabled(SerializationConfig.Feature.INDENT_OUTPUT)) {
            jgen.useDefaultPrettyPrinter();
        }
        // [JACKSON-282]: consider Closeable
        if (cfg.isEnabled(SerializationConfig.Feature.CLOSE_CLOSEABLE) && (value instanceof Closeable)) {
            _configAndWriteCloseable(jgen, value, cfg);
            return;
        }
        boolean closed = false;
        try {
            _serializerProvider.serializeValue(cfg, jgen, value, _serializerFactory);
            closed = true;
            jgen.close();
```

```
    } finally {
        /* won't try to close twice; also, must catch exception (so it
         * will not mask exception that is pending)
         */
        if (!closed) {
            try {
                jgen.close();
            } catch (IOException ioe) { }
        }
    }
}

protected final void _configAndWriteValue(JsonGenerator jgen, Object value, Class<?> viewClass)
    throws IOException, JsonGenerationException, JsonMappingException
{
    SerializationConfig cfg = copySerializationConfig();
    if (cfg.isEnabled(SerializationConfig.Feature.INDENT_OUTPUT)) {
        jgen.useDefaultPrettyPrinter();
    }
    cfg.setSerializationView(viewClass);
    // [JACKSON-282]: consider Closeable
    if (cfg.isEnabled(SerializationConfig.Feature.CLOSE_CLOSEABLE) && (value instanceof Closeable)) {
        _configAndWriteCloseable(jgen, value, cfg);
        return;
    }
    boolean closed = false;
    try {
        _serializerProvider.serializeValue(cfg, jgen, value, _serializerFactory);
        closed = true;
        jgen.close();
    } finally {
        if (!closed) {
            try {
                jgen.close();
            } catch (IOException ioe) { }
        }
    }
}

/**
 * Helper method used when value to serialize is {@link Closeable} and its <code>close()</code>
 * method is to be called right after serialization has been called
 */
private final void _configAndWriteCloseable(JsonGenerator jgen, Object value, SerializationConfig cfg)
    throws IOException, JsonGenerationException, JsonMappingException
{
    Closeable toClose = (Closeable) value;
    try {
```

```
        _serializerProvider.serializeValue(cfg, jgen, value, _serializerFactory);
        JsonGenerator tmpJgen = jgen;
        jgen = null;
        tmpJgen.close();
        Closeable tmpToClose = toClose;
        toClose = null;
        tmpToClose.close();
    } finally {
        /* Need to close both generator and value, as long as they haven't yet
         * been closed
         */
        if (jgen != null) {
            try {
                jgen.close();
            } catch (IOException ioe) { }
        }
        if (toClose != null) {
            try {
                toClose.close();
            } catch (IOException ioe) { }
        }
    }
}

/**
 * Helper method used when value to serialize is {@link Closeable} and its <code>close()</code>
 * method is to be called right after serialization has been called
 */
private final void _writeCloseableValue(JsonGenerator jgen, Object value, SerializationConfig cfg)
    throws IOException, JsonGenerationException, JsonMappingException
{
    Closeable toClose = (Closeable) value;
    try {
        _serializerProvider.serializeValue(cfg, jgen, value, _serializerFactory);
        if (cfg.isEnabled(SerializationConfig.Feature.FLUSH_AFTER_WRITE_VALUE)) {
            jgen.flush();
        }
        Closeable tmpToClose = toClose;
        toClose = null;
        tmpToClose.close();
    } finally {
        if (toClose != null) {
            try {
                toClose.close();
            } catch (IOException ioe) { }
        }
    }
}
```

```
/**
 * Actual implementation of value reading+binding operation.
 */
protected Object _readValue(DeserializationConfig cfg, JsonParser jp, JavaType valueType)
   throws IOException, JsonParseException, JsonMappingException
{
   /* First: may need to read the next token, to initialize
    * state (either before first read from parser, or after
    * previous token has been cleared)
    */
   Object result;
   JsonToken t = _initForReading(jp);
   if (t == JsonToken.VALUE_NULL || t == JsonToken.END_ARRAY || t == JsonToken.END_OBJECT) {
      result = null;
   } else { // pointing to event other than null
      DeserializationContext ctxt = _createDeserializationContext(jp, cfg);
      // ok, let's get the value
      result = _findRootDeserializer(cfg, valueType).deserialize(jp, ctxt);
   }
   // Need to consume the token too
   jp.clearCurrentToken();
   return result;
}


protected Object _readMapAndClose(JsonParser jp, JavaType valueType)
   throws IOException, JsonParseException, JsonMappingException
{
   try {
      Object result;
      JsonToken t = _initForReading(jp);
      if (t == JsonToken.VALUE_NULL || t == JsonToken.END_ARRAY || t == JsonToken.END_OBJECT) {
         result = null;
      } else {
         DeserializationConfig cfg = copyDeserializationConfig();
         DeserializationContext ctxt = _createDeserializationContext(jp, cfg);
         result = _findRootDeserializer(cfg, valueType).deserialize(jp, ctxt);
      }
      // Need to consume the token too
      jp.clearCurrentToken();
      return result;
   } finally {
      try {
         jp.close();
      } catch (IOException ioe) { }
   }
}
```

```
/**
 * Method called to ensure that given parser is ready for reading
 * content for data binding.
 *
 * @return First token to be used for data binding after this call:
 *  can never be null as exception will be thrown if parser can not
 *  provide more tokens.
 *
 * @throws IOException if the underlying input source has problems during
 *   parsing
 * @throws JsonParseException if parser has problems parsing content
 * @throws JsonMappingException if the parser does not have any more
 *   content to map (note: Json "null" value is considered content;
 *   enf-of-stream not)
 */
protected JsonToken _initForReading(JsonParser jp)
   throws IOException, JsonParseException, JsonMappingException
{
   /* First: must point to a token; if not pointing to one, advance.
    * This occurs before first read from JsonParser, as well as
    * after clearing of current token.
    */
   JsonToken t = jp.getCurrentToken();
   if (t == null) {
      // and then we must get something...
      t = jp.nextToken();
      if (t == null) {
         /* [JACKSON-99] Should throw EOFException, closed thing
          *   semantically
          */
         throw new EOFException("No content to map to Object due to end of input");
      }
   }
   return t;
}

/*
/**********************************************************
/* Internal methods, other
/**********************************************************
 */

/**
 * Method called to locate deserializer for the passed root-level value.
 */
protected JsonDeserializer<Object> _findRootDeserializer(DeserializationConfig cfg, JavaType valueType)
   throws JsonMappingException
```

```
  {
    // First: have we already seen it?
    JsonDeserializer<Object> deser = _rootDeserializers.get(valueType);
    if (deser != null) {
      return deser;
    }
    // Nope: need to ask provider to resolve it
    deser = _deserializerProvider.findTypedValueDeserializer(cfg, valueType, null);
    if (deser == null) { // can this happen?
      throw new JsonMappingException("Can not find a deserializer for type "+valueType);
    }
    _rootDeserializers.put(valueType, deser);
    return deser;
  }

  protected DeserializationContext _createDeserializationContext(JsonParser jp, DeserializationConfig cfg)
  {
    // 04-Jan-2010, tatu: we do actually need the provider too... (for polymorphic deser)
    return new StdDeserializationContext(cfg, jp, _deserializerProvider);
  }

  //Allows use of the correct classloader (primarily for OSGi), separating framework from application
  //should be safe to use in all contexts
  /*
  protected <T> void _setupClassLoaderForDeserialization(Class<T> valueType)
  {
    ClassLoader loader = (valueType.getClassLoader() == null) ? Thread.currentThread().getContextClassLoader()
: valueType.getClassLoader();
    Thread.currentThread().setContextClassLoader(loader);
  }
  */
}
package org.codehaus.jackson.map;

import java.io.IOException;

import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonProcessingException;

/**
* Interface that is to replace {@link JsonSerializable} to
* allow for dynamic type information embedding.
*
* @since 1.5
* @author tatu
*/
@SuppressWarnings("deprecation")
public interface JsonSerializableWithType
```

```
  extends JsonSerializable
{
  public void serializeWithType(JsonGenerator jgen, SerializerProvider provider,
        TypeSerializer typeSer)
    throws IOException, JsonProcessingException;
}
/**
* Utility classes for Mapper package.
*/
package org.codehaus.jackson.map.util;
package org.codehaus.jackson.map.util;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.type.JavaType;

/**
* Container class that can be used to wrap any Object instances (including
* nulls), and will serialize embedded in
* <a href="http://en.wikipedia.org/wiki/JSONP">JSONP</a> wrapping.
*
* @see org.codehaus.jackson.map.util.JSONWrappedObject
*
* @author tatu
* @since 1.5
*/
public class JSONPObject
  implements JsonSerializableWithType
{
  /**
   * JSONP function name to use for serialization
   */
  protected final String _function;

  /**
   * Value to be serialized as JSONP padded; can be null.
   */
  protected final Object _value;

  /**
   * Optional static type to use for serialization; if null, runtime
   * type is used. Can be used to specify declared type which defines
   * serializer to use, as well as aspects of extra type information
   * to include (if any).
   */
```

```
protected final JavaType _serializationType;

public JSONPObject(String function, Object value) {
   this(function, value, (JavaType) null);
}

public JSONPObject(String function, Object value, JavaType asType)
{
   _function = function;
   _value = value;
   _serializationType = asType;
}

public JSONPObject(String function, Object value, Class<?> rawType) {
   _function = function;
   _value = value;
   _serializationType = (rawType == null) ? null : TypeFactory.type(rawType);
}

/*
/**********************************************************
/* JsonSerializable(WithType) implementation
/**********************************************************
 */

public void serializeWithType(JsonGenerator jgen, SerializerProvider provider, TypeSerializer typeSer)
       throws IOException, JsonProcessingException
{
   // No type for JSONP wrapping: value serializer will handle typing for value:
   serialize(jgen, provider);
}

public void serialize(JsonGenerator jgen, SerializerProvider provider)
       throws IOException, JsonProcessingException
{
   // First, wrapping:
   jgen.writeRaw(_function);
   jgen.writeRaw('(');
   if (_value == null) {
      provider.defaultSerializeNull(jgen);
   } else if (_serializationType != null) {
      provider.findTypedValueSerializer(_serializationType, true, null).serialize(_value, jgen, provider);
   } else {
      Class<?> cls = _value.getClass();
      provider.findTypedValueSerializer(cls, true, null).serialize(_value, jgen, provider);
   }
   jgen.writeRaw(')');
}
```

```
  /*
  /**********************************************************************
  /* Accessors
  /**********************************************************************
   */

  public String getFunction() { return _function; }
  public Object getValue() { return _value; }
  public JavaType getSerializationType() { return _serializationType; }
}
package org.codehaus.jackson.map.util;

/**
* Node of a forward-only linked list.
*
* @author tatu
*
* @param <T> Type of contained object
*/
public final class LinkedNode<T>
{
  final T _value;
  final LinkedNode<T> _next;

  public LinkedNode(T value, LinkedNode<T> next)
  {
    _value = value;
    _next = next;
  }

  public LinkedNode<T> next() { return _next; }

  public T value() { return _value; }

  /**
   * Convenience method that can be used to check if a linked list
   * with given head node (which may be null to indicate empty list)
   * contains given value
   *
   * @param <ST> Type argument that defines contents of the linked list parameter
   * @param node Head node of the linked list
   * @param value Value to look for
   * @return True if linked list contains the value, false otherwise
   */
  public static <ST> boolean contains(LinkedNode<ST> node, ST value)
  {
    while (node != null) {
```

```
            if (node.value() == value) {
                return true;
            }
            node = node.next();
        }
        return false;
    }
}
package org.codehaus.jackson.map.util;

import java.lang.reflect.Array;
import java.util.List;

/**
* Helper class to use for constructing Object arrays by appending entries
* to create arrays of various lengths (length that is not known a priori).
*/
public final class ObjectBuffer
{
    // // // Config constants

    /**
     * Let's start with small chunks; typical usage is for small arrays anyway.
     */
    final static int INITIAL_CHUNK_SIZE = 12;

    /**
     * Also: let's expand by doubling up until 64k chunks (which is 16k entries for
     * 32-bit machines)
     */
    final static int SMALL_CHUNK_SIZE = (1 << 14);

    /**
     * Let's limit maximum size of chunks we use; helps avoid excessive allocation
     * overhead for huge data sets.
     * For now, let's limit to quarter million entries, 1 meg chunks for 32-bit
     * machines.
     */
    final static int MAX_CHUNK_SIZE = (1 << 18);

    // // // Data storage

    private Node _bufferHead;

    private Node _bufferTail;

    /**
     * Number of total buffered entries in this buffer, counting all instances
```

```
 * within linked list formed by following {@link #_bufferHead}.
 */
private int _bufferedEntryCount;

// // // Simple reuse

/**
 * Reusable Object array, stored here after buffer has been released having
 * been used previously.
 */
private Object[] _freeBuffer;

/*
/**********************************************************
/* Construction
/**********************************************************
 */

public ObjectBuffer() { }

/*
/**********************************************************
/* Public API
/**********************************************************
 */

/**
 * Method called to start buffering process. Will ensure that the buffer
 * is empty, and then return an object array to start chunking content on
 */
public Object[] resetAndStart()
{
   _reset();
   if (_freeBuffer == null) {
      return new Object[INITIAL_CHUNK_SIZE];
   }
   return _freeBuffer;
}

/**
 * Method called to add a full Object array as a chunk buffered within
 * this buffer, and to obtain a new array to fill. Caller is not to use
 * the array it gives; but to use the returned array for continued
 * buffering.
 *
 * @param fullChunk Completed chunk that the caller is requesting
 *   to append to this buffer. It is generally chunk that was
 *   returned by an earlier call to {@link #resetAndStart} or
```

```
 *   {@link #appendCompletedChunk} (although this is not required or
 *   enforced)
 *
 * @return New chunk buffer for caller to fill
 */
public Object[] appendCompletedChunk(Object[] fullChunk)
{
   Node next = new Node(fullChunk);
   if (_bufferHead == null) { // first chunk
      _bufferHead = _bufferTail = next;
   } else { // have something already
      _bufferTail.linkNext(next);
      _bufferTail = next;
   }
   int len = fullChunk.length;
   _bufferedEntryCount += len;
   // double the size for small chunks
   if (len < SMALL_CHUNK_SIZE) {
      len += len;
   } else { // but by +25% for larger (to limit overhead)
      len += (len >> 2);
   }
   return new Object[len];
}

/**
 * Method called to indicate that the buffering process is now
 * complete; and to construct a combined exactly-sized result
 * array. Additionally the buffer itself will be reset to
 * reduce memory retention.
 *<p>
 * Resulting array will be of generic <code>Object[]</code> type:
 * if a typed array is needed, use the method with additional
 * type argument.
 */
public Object[] completeAndClearBuffer(Object[] lastChunk, int lastChunkEntries)
{
   int totalSize = lastChunkEntries + _bufferedEntryCount;
   Object[] result = new Object[totalSize];
   _copyTo(result, totalSize, lastChunk, lastChunkEntries);
   return result;
}

/**
 * Type-safe alternative to
 * {@link #completeAndClearBuffer(Object[], int)}, to allow
 * for constructing explicitly typed result array.
 *
```

```
 * @param componentType Type of elements included in the buffer. Will be
 *   used for constructing the result array.
 */
public <T> T[] completeAndClearBuffer(Object[] lastChunk, int lastChunkEntries, Class<T> componentType)
{
  int totalSize = lastChunkEntries + _bufferedEntryCount;
 @SuppressWarnings("unchecked")
   T[] result = (T[]) Array.newInstance(componentType, totalSize);
   _copyTo(result, totalSize, lastChunk, lastChunkEntries);
   _reset();
   return result;
}

/**
 * Another
 *
 * @since 1.6
 */
public void completeAndClearBuffer(Object[] lastChunk, int lastChunkEntries, List<Object> resultList)
{
   for (Node n = _bufferHead; n != null; n = n.next()) {
      Object[] curr = n.getData();
      for (int i = 0, len = curr.length; i < len; ++i) {
         resultList.add(curr[i]);
      }
   }
   // and then the last one
   for (int i = 0; i < lastChunkEntries; ++i) {
      resultList.add(lastChunk[i]);
   }
}

/**
 * Helper method that can be used to check how much free capacity
 * will this instance start with. Can be used to choose the best
 * instance to reuse, based on size of reusable object chunk
 * buffer holds reference to.
 */
public int initialCapacity()
{
   return (_freeBuffer == null) ? 0 : _freeBuffer.length;
}

/**
 * Method that can be used to check how many Objects have been buffered
 * within this buffer.
 */
public int bufferedSize() { return _bufferedEntryCount; }
```

```
/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

protected void _reset()
{
   // can we reuse the last (and thereby biggest) array for next time?
   if (_bufferTail != null) {
      _freeBuffer = _bufferTail.getData();
   }
   // either way, must discard current contents
   _bufferHead = _bufferTail = null;
   _bufferedEntryCount = 0;
}

protected final void _copyTo(Object resultArray, int totalSize,
                             Object[] lastChunk, int lastChunkEntries)
{
   int ptr = 0;

   for (Node n = _bufferHead; n != null; n = n.next()) {
      Object[] curr = n.getData();
      int len = curr.length;
      System.arraycopy(curr, 0, resultArray, ptr, len);
      ptr += len;
   }
   System.arraycopy(lastChunk, 0, resultArray, ptr, lastChunkEntries);
   ptr += lastChunkEntries;

   // sanity check (could have failed earlier due to out-of-bounds, too)
   if (ptr != totalSize) {
      throw new IllegalStateException("Should have gotten "+totalSize+" entries, got "+ptr);
   }
}

/*
/**********************************************************
/* Helper classes
/**********************************************************
 */

/**
 * Helper class used to store actual data, in a linked list.
 */
final static class Node
```

```
  {
    /**
     * Data stored in this node. Array is considered to be full.
     */
    final Object[] _data;

    Node _next;

    public Node(Object[] data) {
      _data = data;
    }

    public Object[] getData() { return _data; }

    public Node next() { return _next; }

    public void linkNext(Node next)
    {
      if (_next != null) { // sanity check
        throw new IllegalStateException();
      }
      _next = next;
    }
  }
}
package org.codehaus.jackson.map.util;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.type.JavaType;


/**
* General-purpose wrapper class that can be used to decorate serialized
* value with arbitrary literal prefix and suffix. This can be used for
* example to construct arbitrary Javascript values (similar to how basic
* function name and parenthesis are used with JSONP).
*
* @see org.codehaus.jackson.map.util.JSONPObject
*
* @author tatu
* @since 1.5
*/
public class JSONWrappedObject
  implements JsonSerializableWithType
```

```
{
  /**
   * Literal String to output before serialized value.
   * Will not be quoted when serializing value.
   */
  protected final String _prefix;

  /**
   * Literal String to output after serialized value.
   * Will not be quoted when serializing value.
   */
  protected final String _suffix;

  /**
   * Value to be serialized as JSONP padded; can be null.
   */
  protected final Object _value;

  /**
   * Optional static type to use for serialization; if null, runtime
   * type is used. Can be used to specify declared type which defines
   * serializer to use, as well as aspects of extra type information
   * to include (if any).
   */
  protected final JavaType _serializationType;

  public JSONWrappedObject(String prefix, String suffix, Object value) {
     this(prefix, suffix, value, (JavaType) null);
  }

  public JSONWrappedObject(String prefix, String suffix, Object value, JavaType asType)
  {
     _prefix = prefix;
     _suffix = suffix;
     _value = value;
     _serializationType = asType;
  }

  public JSONWrappedObject(String prefix, String suffix, Object value, Class<?> rawType) {
     _prefix = prefix;
     _suffix = suffix;
     _value = value;
     _serializationType = (rawType == null) ? null : TypeFactory.type(rawType);
  }

  /*
  /**********************************************************
  /* JsonSerializable(WithType) implementation
```

```
    /**********************************************************
     */

    public void serializeWithType(JsonGenerator jgen, SerializerProvider provider, TypeSerializer typeSer)
            throws IOException, JsonProcessingException
    {
        // No type for JSONP wrapping: value serializer will handle typing for value:
        serialize(jgen, provider);
    }

    public void serialize(JsonGenerator jgen, SerializerProvider provider)
            throws IOException, JsonProcessingException
    {
        // First, wrapping:
     if (_prefix != null) jgen.writeRaw(_prefix);
        if (_value == null) {
            provider.defaultSerializeNull(jgen);
        } else if (_serializationType != null) {
            provider.findTypedValueSerializer(_serializationType, true, null).serialize(_value, jgen, provider);
        } else {
            Class<?> cls = _value.getClass();
            provider.findTypedValueSerializer(cls, true, null).serialize(_value, jgen, provider);
        }
        if (_suffix != null) jgen.writeRaw(_suffix);
    }

    /*
    /**********************************************************
    /* Accessors
    /**********************************************************
     */

    public String getPrefix() { return _prefix; }
    public String getSuffix() { return _suffix; }
    public Object getValue() { return _value; }
    public JavaType getSerializationType() { return _serializationType; }

}
package org.codehaus.jackson.map.util;

import java.text.DateFormat;
import java.text.FieldPosition;
import java.text.ParseException;
import java.text.ParsePosition;
import java.text.SimpleDateFormat;
import java.util.*;

import org.codehaus.jackson.io.NumberInput;
```

```
/**
* Default {@link DateFormat} implementation used by standard Date
* serializers and deserializers. For serialization defaults to using
* an ISO-8601 compliant format (format String "yyyy-MM-dd'T'HH:mm:ss.SSSZ")
* and for deserialization, both ISO-8601 and RFC-1123.
*/
@SuppressWarnings("serial")
public class StdDateFormat
  extends DateFormat
{
  /* TODO !!! 24-Nov-2009, tatu: Need to rewrite this class soon:
   * JDK date parsing is awfully brittle, and ISO-8601 is quite
   * permissive. The two don't mix, need to write a better one.
   */

  /**
   * Defines a commonly used date format that conforms
   * to ISO-8601 date formatting standard, when it includes basic undecorated
   * timezone definition
   */
  final static String DATE_FORMAT_STR_ISO8601 = "yyyy-MM-dd'T'HH:mm:ss.SSSZ";

  /**
   * Same as 'regular' 8601, but handles 'Z' as an alias for "+0000"
   * (or "GMT")
   */
  final static String DATE_FORMAT_STR_ISO8601_Z = "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'";

  /**
   * ISO-8601 with just the Date part, no time
   *
   * @since 1.3.1
   */
  final static String DATE_FORMAT_STR_PLAIN = "yyyy-MM-dd";

  /**
   * This constant defines the date format specified by
   * RFC 1123.
   */
  final static String DATE_FORMAT_STR_RFC1123 = "EEE, dd MMM yyyy HH:mm:ss zzz";

  /**
   * For error messages we'll also need a list of all formats.
   */
  final static String[] ALL_FORMATS = new String[] {
     DATE_FORMAT_STR_ISO8601,
     DATE_FORMAT_STR_ISO8601_Z,
```

```
    DATE_FORMAT_STR_RFC1123,
    DATE_FORMAT_STR_PLAIN
};

final static SimpleDateFormat DATE_FORMAT_RFC1123;

final static SimpleDateFormat DATE_FORMAT_ISO8601;
final static SimpleDateFormat DATE_FORMAT_ISO8601_Z;

final static SimpleDateFormat DATE_FORMAT_PLAIN;

/* Let's construct "blueprint" date format instances: can not be used
 * as is, due to thread-safety issues, but can be used for constructing
 * actual instances more cheaply (avoids re-parsing).
 */
static {
    /* Another important thing: let's force use of GMT for
     * baseline DataFormat objects
     */
    TimeZone gmt = TimeZone.getTimeZone("GMT");
    DATE_FORMAT_RFC1123 = new SimpleDateFormat(DATE_FORMAT_STR_RFC1123);
    DATE_FORMAT_RFC1123.setTimeZone(gmt);
    DATE_FORMAT_ISO8601 = new SimpleDateFormat(DATE_FORMAT_STR_ISO8601);
    DATE_FORMAT_ISO8601.setTimeZone(gmt);
    DATE_FORMAT_ISO8601_Z = new SimpleDateFormat(DATE_FORMAT_STR_ISO8601_Z);
    DATE_FORMAT_ISO8601_Z.setTimeZone(gmt);
    DATE_FORMAT_PLAIN = new SimpleDateFormat(DATE_FORMAT_STR_PLAIN);
    DATE_FORMAT_PLAIN.setTimeZone(gmt);
}

/**
 * A singleton instance can be used for cloning purposes.
 */
public final static StdDateFormat instance = new StdDateFormat();

transient SimpleDateFormat _formatRFC1123;
transient SimpleDateFormat _formatISO8601;
transient SimpleDateFormat _formatISO8601_z;
transient SimpleDateFormat _formatPlain;

/*
/**********************************************************
/* Life cycle, accessing singleton "standard" formats
/**********************************************************
 */

public StdDateFormat() { }
```

```
@Override
public StdDateFormat clone() {
    /* Since we always delegate all work to child DateFormat instances,
     * let's NOT call super.clone(); this is bit unusual, but makes
     * sense here to avoid unnecessary work.
     */
    return new StdDateFormat();
}

/**
 * Method for getting the globally shared DateFormat instance
 * that uses GMT timezone and can handle simple ISO-8601
 * compliant date format.
 */
public static DateFormat getBlueprintISO8601Format() {
    return DATE_FORMAT_ISO8601;
}

/**
 * Method for getting a non-shared DateFormat instance
 * that uses specified timezone and can handle simple ISO-8601
 * compliant date format.
 */
public static DateFormat getISO8601Format(TimeZone tz) {
    DateFormat df = (SimpleDateFormat) DATE_FORMAT_ISO8601.clone();
    df.setTimeZone(tz);
    return df;
}

/**
 * Method for getting the globally shared DateFormat instance
 * that uses GMT timezone and can handle RFC-1123
 * compliant date format.
 */
public static DateFormat getBlueprintRFC1123Format() {
    return DATE_FORMAT_RFC1123;
}


/**
 * Method for getting a non-shared DateFormat instance
 * that uses specific timezone and can handle RFC-1123
 * compliant date format.
 */
public static DateFormat getRFC1123Format(TimeZone tz)
{
    DateFormat df = (SimpleDateFormat) DATE_FORMAT_RFC1123.clone();
    df.setTimeZone(tz);
```

```
    return df;
}

/*
/**********************************************************
/* Public API
/**********************************************************
 */

@Override
public Date parse(String dateStr) throws ParseException
{
    dateStr = dateStr.trim();
    ParsePosition pos = new ParsePosition(0);
    Date result = parse(dateStr, pos);
    if (result != null) {
        return result;
    }

    StringBuilder sb = new StringBuilder();
    for (String f : ALL_FORMATS) {
        if (sb.length() > 0) {
            sb.append("\", \"");
        } else {
            sb.append("");
        }
        sb.append(f);
    }
    sb.append("");
    throw new ParseException
        (String.format("Can not parse date \"%s\": not compatible with any of standard forms (%s)",
                dateStr, sb.toString()), pos.getErrorIndex());
}

@Override
public Date parse(String dateStr, ParsePosition pos)
{
    if (looksLikeISO8601(dateStr)) { // also includes "plain"
        return parseAsISO8601(dateStr, pos);
    }
    /* 14-Feb-2010, tatu: As per [JACKSON-236], better also
     *   consider "stringified" simple time stamp
     */
    int i = dateStr.length();
    while (--i >= 0) {
        char ch = dateStr.charAt(i);
        if (ch < '0' || ch > '9') break;
    }
```

```
    if (i < 0) { // all digits
        if (NumberInput.inLongRange(dateStr, false)) {
            return new Date(Long.parseLong(dateStr));
        }
    }
    // Otherwise, fall back to using RFC 1123
    return parseAsRFC1123(dateStr, pos);
}

@Override
public StringBuffer format(Date date, StringBuffer toAppendTo,
                           FieldPosition fieldPosition)
{
    if (_formatISO8601 == null) {
        _formatISO8601 = (SimpleDateFormat) DATE_FORMAT_ISO8601.clone();
    }
    return _formatISO8601.format(date, toAppendTo, fieldPosition);
}

/*
/**********************************************************
/* Helper methods
/**********************************************************
 */

/**
 * Overridable helper method used to figure out which of supported
 * formats is the likeliest match.
 */
protected boolean looksLikeISO8601(String dateStr)
{
    if (dateStr.length() >= 5
        && Character.isDigit(dateStr.charAt(0))
        && Character.isDigit(dateStr.charAt(3))
        && dateStr.charAt(4) == '-'
        ) {
        return true;
    }
    return false;
}

protected Date parseAsISO8601(String dateStr, ParsePosition pos)
{
    /* 21-May-2009, tatu: SimpleDateFormat has very strict handling of
     * timezone  modifiers for ISO-8601. So we need to do some scrubbing.
     */

    /* First: do we have "zulu" format ('Z' == "GMT")? If yes, that's
```

```
 * quite simple because we already set date format timezone to be
 * GMT, and hence can just strip out 'Z' altogether
 */
int len = dateStr.length();
char c = dateStr.charAt(len-1);
SimpleDateFormat df;

// [JACKSON-200]: need to support "plain" date...
if (len <= 10 && Character.isDigit(c)) {
  df = _formatPlain;
   if (df == null) {
      df = _formatPlain = (SimpleDateFormat) DATE_FORMAT_PLAIN.clone();
   }
} else if (c == 'Z') {
   df = _formatISO8601_z;
   if (df == null) {
      df = _formatISO8601_z = (SimpleDateFormat) DATE_FORMAT_ISO8601_Z.clone();
   }
   // [JACKSON-334]: may be missing milliseconds... if so, add
   if (dateStr.charAt(len-4) == ':') {
      StringBuilder sb = new StringBuilder(dateStr);
      sb.insert(len-1, ".000");
      dateStr = sb.toString();
   }
} else {
   // Let's see if we have timezone indicator or not...
   if (hasTimeZone(dateStr)) {
      c = dateStr.charAt(len-3);
      if (c == ':') { // remove optional colon
         // remove colon
         StringBuilder sb = new StringBuilder(dateStr);
         sb.delete(len-3, len-2);
         dateStr = sb.toString();
      } else if (c == '+' || c == '-') { // missing minutes
         // let's just append '00'
         dateStr += "00";
      }
      // [JACKSON-334]: may be missing milliseconds... if so, add
      len = dateStr.length();
      // '+0000' (5 chars); should come after '.000' (4 chars) of milliseconds, so:
      c = dateStr.charAt(len-9);
      if (Character.isDigit(c)) {
         StringBuilder sb = new StringBuilder(dateStr);
         sb.insert(len-5, ".000");
         dateStr = sb.toString();
      }

      df = _formatISO8601;
```

```
            if (_formatISO8601 == null) {
                df = _formatISO8601 = (SimpleDateFormat) DATE_FORMAT_ISO8601.clone();
            }
        } else {
            /* 24-Nov-2009, tatu: Ugh. This is getting pretty
             *   ugly. Need to rewrite soon!
             */

            // If not, plain date. Easiest to just patch 'Z' in the end?
            StringBuilder sb = new StringBuilder(dateStr);
            // And possible also millisecond part if missing
            int timeLen = len - dateStr.lastIndexOf('T') - 1;
            if (timeLen <= 8) {
                sb.append(".000");
            }
            sb.append('Z');
            dateStr = sb.toString();
            df = _formatISO8601_z;
            if (df == null) {
                df = _formatISO8601_z = (SimpleDateFormat) DATE_FORMAT_ISO8601_Z.clone();
            }
        }
    }
    return df.parse(dateStr, pos);
}

protected Date parseAsRFC1123(String dateStr, ParsePosition pos)
{
    if (_formatRFC1123 == null) {
        _formatRFC1123 = (SimpleDateFormat) DATE_FORMAT_RFC1123.clone();
    }
    return _formatRFC1123.parse(dateStr, pos);
}

private final static boolean hasTimeZone(String str)
{
    // Only accept "+hh", "+hhmm" and "+hh:mm" (and with minus), so
    int len = str.length();
    if (len >= 6) {
        char c = str.charAt(len-6);
        if (c == '+' || c == '-') return true;
        c = str.charAt(len-5);
        if (c == '+' || c == '-') return true;
        c = str.charAt(len-3);
        if (c == '+' || c == '-') return true;
    }
    return false;
}
```

```
}
package org.codehaus.jackson.map.util;

import java.util.*;

/**
* Simple helper class used for decoupling instantiation of
* optionally loaded handlers, like deserializers and deserializers
* for libraries that are only present on some platforms.
*
* @author tatu
*
* @param <T> Type of objects provided
*/
public interface Provider<T>
{
  /**
   * Method used to request provider to provide entries it has
   */
  public Collection<T> provide();
}
package org.codehaus.jackson.map.util;

/**
* Base class for specialized primitive array builders.
*/
public abstract class PrimitiveArrayBuilder<T>
{
  /**
   * Let's start with small chunks; typical usage is for small arrays anyway.
   */
  final static int INITIAL_CHUNK_SIZE = 12;

  /**
   * Also: let's expand by doubling up until 64k chunks (which is 16k entries for
   * 32-bit machines)
   */
  final static int SMALL_CHUNK_SIZE = (1 << 14);

  /**
   * Let's limit maximum size of chunks we use; helps avoid excessive allocation
   * overhead for huge data sets.
   * For now, let's limit to quarter million entries, 1 meg chunks for 32-bit
   * machines.
   */
  final static int MAX_CHUNK_SIZE = (1 << 18);

  // // // Data storage
```

```
T _freeBuffer;

Node<T> _bufferHead;

Node<T> _bufferTail;

/**
 * Number of total buffered entries in this buffer, counting all instances
 * within linked list formed by following {@link #_bufferHead}.
 */
int _bufferedEntryCount;

// // // Recycled instances of sub-classes

// // // Life-cycle

protected PrimitiveArrayBuilder() { }

/*
////////////////////////////////////////////////////////////////////////
// Public API
////////////////////////////////////////////////////////////////////////
 */

public T resetAndStart()
{
    _reset();
    return (_freeBuffer == null) ?
        _constructArray(INITIAL_CHUNK_SIZE) : _freeBuffer;
}

/**
 * @return Length of the next chunk to allocate
 */
public final T appendCompletedChunk(T fullChunk, int fullChunkLength)
{
    Node<T> next = new Node<T>(fullChunk, fullChunkLength);
    if (_bufferHead == null) { // first chunk
        _bufferHead = _bufferTail = next;
    } else { // have something already
        _bufferTail.linkNext(next);
        _bufferTail = next;
    }
    _bufferedEntryCount += fullChunkLength;
    int nextLen = fullChunkLength; // start with last chunk size
    // double the size for small chunks
    if (nextLen < SMALL_CHUNK_SIZE) {
```

```
            nextLen += nextLen;
        } else { // but by +25% for larger (to limit overhead)
            nextLen += (nextLen >> 2);
        }
        return _constructArray(nextLen);
    }

    public T completeAndClearBuffer(T lastChunk, int lastChunkEntries)
    {
        int totalSize = lastChunkEntries + _bufferedEntryCount;
        T resultArray = _constructArray(totalSize);

        int ptr = 0;

        for (Node<T> n = _bufferHead; n != null; n = n.next()) {
            ptr = n.copyData(resultArray, ptr);
        }
        System.arraycopy(lastChunk, 0, resultArray, ptr, lastChunkEntries);
        ptr += lastChunkEntries;

        // sanity check (could have failed earlier due to out-of-bounds, too)
        if (ptr != totalSize) {
            throw new IllegalStateException("Should have gotten "+totalSize+" entries, got "+ptr);
        }
        return resultArray;
    }

    /*
    ////////////////////////////////////////////////////////////////////////
    // Abstract methods for sub-classes to implement
    ////////////////////////////////////////////////////////////////////////
     */

    protected abstract T _constructArray(int len);

    /*
    ////////////////////////////////////////////////////////////////////////
    // Internal methods
    ////////////////////////////////////////////////////////////////////////
     */

    protected void _reset()
    {
        // can we reuse the last (and thereby biggest) array for next time?
        if (_bufferTail != null) {
            _freeBuffer = _bufferTail.getData();
        }
        // either way, must discard current contents
```

```
    _bufferHead = _bufferTail = null;
    _bufferedEntryCount = 0;
}

/*
////////////////////////////////////////////////////////////////////////
// Helper classes
////////////////////////////////////////////////////////////////////////
 */

/**
 * For actual buffering beyond the current buffer, we can actually
 * use shared class which only deals with opaque "untyped" chunks.
 * This works because {@link java.lang.System#arraycopy} does not
 * take type; hence we can implement some aspects of primitive data
 * handling in generic fashion.
 */
final static class Node<T>
{
    /**
     * Data stored in this node.
     */
    final T _data;

    /**
     * Number entries in the (untyped) array. Offset is assumed to be 0.
     */
    final int _dataLength;

    Node<T> _next;

    public Node(T data, int dataLen)
    {
        _data = data;
        _dataLength = dataLen;
    }

    public T getData() { return _data; }

    public int copyData(T dst, int ptr)
    {
        System.arraycopy(_data, 0, dst, ptr, _dataLength);
        ptr += _dataLength;
        return ptr;
    }

    public Node<T> next() { return _next; }
```

```
        public void linkNext(Node<T> next)
        {
            if (_next != null) { // sanity check
                throw new IllegalStateException();
            }
            _next = next;
        }
    }
}
package org.codehaus.jackson.map.util;

import org.codehaus.jackson.io.SerializedString;
import org.codehaus.jackson.map.AnnotationIntrospector;
import org.codehaus.jackson.map.MapperConfig;
import org.codehaus.jackson.map.introspect.AnnotatedClass;
import org.codehaus.jackson.map.introspect.BasicBeanDescription;
import org.codehaus.jackson.map.type.ClassKey;
import org.codehaus.jackson.type.JavaType;

/**
* Helper class for caching resolved root names.
*/
public class RootNameLookup
{
    /**
     * For efficient operation, let's try to minimize number of times we
     * need to introspect root element name to use.
     */
    protected LRUMap<ClassKey,SerializedString> _rootNames;

    public RootNameLookup() { }

    public SerializedString findRootName(JavaType rootType, MapperConfig<?> config)
    {
        return findRootName(rootType.getRawClass(), config);
    }

    public synchronized SerializedString findRootName(Class<?> rootType, MapperConfig<?> config)
    {
        ClassKey key = new ClassKey(rootType);

        if (_rootNames == null) {
            _rootNames = new LRUMap<ClassKey,SerializedString>(20, 200);
        } else {
            SerializedString name = _rootNames.get(key);
            if (name != null) {
                return name;
            }
```

```
        }
        BasicBeanDescription beanDesc = (BasicBeanDescription) config.introspectClassAnnotations(rootType);
        AnnotationIntrospector intr = config.getAnnotationIntrospector();
        AnnotatedClass ac = beanDesc.getClassInfo();
        String nameStr = intr.findRootName(ac);
        // No answer so far? Let's just default to using simple class name
        if (nameStr == null) {
            // Should we strip out enclosing class tho? For now, nope:
            nameStr = rootType.getSimpleName();
        }
        SerializedString name = new SerializedString(nameStr);
        _rootNames.put(key, name);
        return name;
    }
}
package org.codehaus.jackson.map.util;

import java.util.LinkedHashMap;
import java.util.Map;

/**
* Helper for simple bounded LRU maps used for reusing lookup values.
*
* @since 1.7
*/
@SuppressWarnings("serial")
public class LRUMap<K,V> extends LinkedHashMap<K,V>
{
    protected final int _maxEntries;

    public LRUMap(int initialEntries, int maxEntries)
    {
        super(initialEntries, 0.8f, true);
        _maxEntries = maxEntries;
    }

    @Override
    protected boolean removeEldestEntry(Map.Entry<K,V> eldest)
    {
        return size() > _maxEntries;
    }

}
package org.codehaus.jackson.map.util;

import java.lang.reflect.Array;
import java.util.*;
```

```
/**
* Helper class that contains set of distinct builders for different
* arrays of primitive values. It also provides trivially simple
* reuse scheme, which assumes that caller knows not to use instances
* concurrently (which works ok with primitive arrays since they can
* not contain other non-primitive types).
*/
public final class ArrayBuilders
{
  BooleanBuilder _booleanBuilder = null;

  // note: no need for char[] builder, assume they are Strings

  ByteBuilder _byteBuilder = null;
  ShortBuilder _shortBuilder = null;
  IntBuilder _intBuilder = null;
  LongBuilder _longBuilder = null;

  FloatBuilder _floatBuilder = null;
  DoubleBuilder _doubleBuilder = null;

  public ArrayBuilders() { }

  public BooleanBuilder getBooleanBuilder()
  {
     if (_booleanBuilder == null) {
        _booleanBuilder = new BooleanBuilder();
     }
     return _booleanBuilder;
  }

  public ByteBuilder getByteBuilder()
  {
     if (_byteBuilder == null) {
        _byteBuilder = new ByteBuilder();
     }
     return _byteBuilder;
  }
  public ShortBuilder getShortBuilder()
  {
     if (_shortBuilder == null) {
        _shortBuilder = new ShortBuilder();
     }
     return _shortBuilder;
  }
  public IntBuilder getIntBuilder()
  {
     if (_intBuilder == null) {
```

```
        _intBuilder = new IntBuilder();
    }
    return _intBuilder;
}
public LongBuilder getLongBuilder()
{
    if (_longBuilder == null) {
        _longBuilder = new LongBuilder();
    }
    return _longBuilder;
}

public FloatBuilder getFloatBuilder()
{
    if (_floatBuilder == null) {
        _floatBuilder = new FloatBuilder();
    }
    return _floatBuilder;
}
public DoubleBuilder getDoubleBuilder()
{
    if (_doubleBuilder == null) {
        _doubleBuilder = new DoubleBuilder();
    }
    return _doubleBuilder;
}

/*
/**********************************************************
/* Impl classes
/**********************************************************
 */

public final static class BooleanBuilder
    extends PrimitiveArrayBuilder<boolean[]>
{
    public BooleanBuilder() { }
    @Override
    public final boolean[] _constructArray(int len) { return new boolean[len]; }
}

public final static class ByteBuilder
    extends PrimitiveArrayBuilder<byte[]>
{
    public ByteBuilder() { }
    @Override
    public final byte[] _constructArray(int len) { return new byte[len]; }
}
```

```
public final static class ShortBuilder
   extends PrimitiveArrayBuilder<short[]>
{
   public ShortBuilder() { }
   @Override
   public final short[] _constructArray(int len) { return new short[len]; }
}
public final static class IntBuilder
   extends PrimitiveArrayBuilder<int[]>
{
   public IntBuilder() { }
   @Override
   public final int[] _constructArray(int len) { return new int[len]; }
}
public final static class LongBuilder
   extends PrimitiveArrayBuilder<long[]>
{
   public LongBuilder() { }
   @Override
   public final long[] _constructArray(int len) { return new long[len]; }
}

public final static class FloatBuilder
   extends PrimitiveArrayBuilder<float[]>
{
   public FloatBuilder() { }
   @Override
   public final float[] _constructArray(int len) { return new float[len]; }
}
public final static class DoubleBuilder
   extends PrimitiveArrayBuilder<double[]>
{
   public DoubleBuilder() { }
   @Override
   public final double[] _constructArray(int len) { return new double[len]; }
}

/*
/**********************************************************
/* Static helper methods
/**********************************************************
 */

public static <T> HashSet<T> arrayToSet(T[] elements)
{
   HashSet<T> result = new HashSet<T>();
   if (elements != null) {
      for (T elem : elements) {
```

```
            result.add(elem);
        }
    }
    return result;
}

/**
 * Helper method for adding specified element to a List, but also
 * considering case where the List may not have been yet constructed
 * (that is, null is passed instead).
 *
 * @param list List to add to; may be null to indicate that a new
 *    List is to be constructed
 * @param element Element to add to list
 *
 * @return List in which element was added; either <code>list</code>
 *   (if it was not null), or a newly constructed List.
 */
public static <T> List<T> addToList(List<T> list, T element)
{
    if (list == null) {
        list = new ArrayList<T>();
    }
    list.add(element);
    return list;
}

/**
 * Helper method for constructing a new array that contains specified
 * element followed by contents of the given array
 */
public static <T> T[] insertInList(T[] array, T element)
{
    int len = array.length;
    @SuppressWarnings("unchecked")
    T[] result = (T[]) Array.newInstance(array.getClass().getComponentType(), len+1);
    if (len > 0) {
        System.arraycopy(array, 0, result, 1, len);
    }
    result[0] = element;
    return result;
}

/**
 * Helper method for exposing contents of arrays using a read-only
 * iterator
 *
 * @since 1.7
```

```
 */
public static <T> Iterator<T> arrayAsIterator(T[] array)
{
   return new ArrayIterator<T>(array);
}

public static <T> Iterable<T> arrayAsIterable(T[] array)
{
   return new ArrayIterator<T>(array);
}

/*
/**********************************************************
/* Helper classes
/**********************************************************
 */

/**
 * Iterator implementation used to efficiently expose contents of an
 * Array as read-only iterator.
 *
 * @since 1.7
 */
private final static class ArrayIterator<T>
   implements Iterator<T>, Iterable<T>
{
   private final T[] _array;

   private int _index;

   public ArrayIterator(T[] array) {
      _array = array;
      _index = 0;
   }

   @Override public boolean hasNext() {
      return _index < _array.length;
   }

   @Override
   public T next()
   {
      if (_index >= _array.length) {
         throw new NoSuchElementException();
      }
      return _array[_index++];
   }
```

```
    @Override public void remove() {
      throw new UnsupportedOperationException();
    }

    @Override
    public Iterator<T> iterator() {
      return this;
    }
  }

}
package org.codehaus.jackson.map.util;

import java.util.*;

import org.codehaus.jackson.io.SerializedString;
import org.codehaus.jackson.map.*;

/**
* Helper class used for storing String serializations of
* enumerations.
*/
public final class EnumValues
{
  /**
   * Since 1.7, we are storing values as SerializedStrings, to further
   * speed up serialization.
   */
  private final EnumMap<?,SerializedString> _values;

  @SuppressWarnings("unchecked")
  private EnumValues(Map<Enum<?>,SerializedString> v) {
    _values = new EnumMap(v);
  }

  public static EnumValues construct(Class<Enum<?>> enumClass, AnnotationIntrospector intr)
  {
    return constructFromName(enumClass, intr);
  }

  public static EnumValues constructFromName(Class<Enum<?>> enumClass, AnnotationIntrospector intr)
  {
    /* [JACKSON-214]: Enum types with per-instance sub-classes
     *   need special handling
     */
   Class<? extends Enum<?>> cls = ClassUtil.findEnumType(enumClass);
    Enum<?>[] values = cls.getEnumConstants();
    if (values != null) {
```

```
            // Type juggling... unfortunate
            Map<Enum<?>,SerializedString> map = new HashMap<Enum<?>,SerializedString>();
            for (Enum<?> en : values) {
                String value = intr.findEnumValue(en);
                map.put(en, new SerializedString(value));
            }
            return new EnumValues(map);
        }
        throw new IllegalArgumentException("Can not determine enum constants for Class "+enumClass.getName());
    }

    public static EnumValues constructFromToString(Class<Enum<?>> enumClass, AnnotationIntrospector intr)
    {
        Class<? extends Enum<?>> cls = ClassUtil.findEnumType(enumClass);
        Enum<?>[] values = cls.getEnumConstants();
        if (values != null) {
            // Type juggling... unfortunate
            Map<Enum<?>,SerializedString> map = new HashMap<Enum<?>,SerializedString>();
            for (Enum<?> en : values) {
                map.put(en, new SerializedString(en.toString()));
            }
            return new EnumValues(map);
        }
        throw new IllegalArgumentException("Can not determine enum constants for Class "+enumClass.getName());
    }

    /**
     * @deprecated since 1.7, use {@link #serializedValueFor} instead
     */
    @Deprecated
    public String valueFor(Enum<?> key)
    {
        SerializedString sstr = _values.get(key);
        return (sstr == null) ? null : sstr.getValue();
    }

    public SerializedString serializedValueFor(Enum<?> key)
    {
        return _values.get(key);
    }

    public Collection<SerializedString> values() {
        return _values.values();
    }
}
package org.codehaus.jackson.map.util;

import java.lang.reflect.*;
```

```
import java.util.*;

public final class ClassUtil
{
  /*
  /**********************************************************
  /* Methods that deal with inheritance
  /**********************************************************
   */

  /**
   * Method that will find all sub-classes and implemented interfaces
   * of a given class or interface. Classes are listed in order of
   * precedence, starting with the immediate super-class, followed by
   * interfaces class directly declares to implemented, and then recursively
   * followed by parent of super-class and so forth.
   * Note that <code>Object.class</code> is not included in the list
   * regardless of whether <code>endBefore</code> argument is defined or not.
   *
   * @param endBefore Super-type to NOT include in results, if any; when
   *    encountered, will be ignored (and no super types are checked).
   */
  public static List<Class<?>> findSuperTypes(Class<?> cls, Class<?> endBefore)
  {
    return findSuperTypes(cls, endBefore, new ArrayList<Class<?>>());
  }

  public static List<Class<?>> findSuperTypes(Class<?> cls, Class<?> endBefore, List<Class<?>> result)
  {
    _addSuperTypes(cls, endBefore, result, false);
    return result;
  }

  private static void _addSuperTypes(Class<?> cls, Class<?> endBefore, Collection<Class<?>> result, boolean
addClassItself)
  {
    if (cls == endBefore || cls == null || cls == Object.class) {
      return;
    }
    if (addClassItself) {
      if (result.contains(cls)) { // already added, no need to check supers
        return;
      }
      result.add(cls);
    }
    for (Class<?> intCls : cls.getInterfaces()) {
      _addSuperTypes(intCls, endBefore, result, true);
    }
```

```
    _addSuperTypes(cls.getSuperclass(), endBefore, result, true);
}

/*
/**********************************************************
/* Class type detection methods
/**********************************************************
 */

/**
 * @return Null if class might be a bean; type String (that identifies
 *   why it's not a bean) if not
 */
public static String canBeABeanType(Class<?> type)
{
    // First: language constructs that ain't beans:
    if (type.isAnnotation()) {
        return "annotation";
    }
    if (type.isArray()) {
        return "array";
    }
    if (type.isEnum()) {
        return "enum";
    }
    if (type.isPrimitive()) {
        return "primitive";
    }

    // Anything else? Seems valid, then
    return null;
}

public static String isLocalType(Class<?> type)
{
    /* As per [JACKSON-187], GAE seems to throw SecurityExceptions
     * here and there... and GAE itself has a bug, too
     * (see []). Bah.
     */
    try {
        // one more: method locals, anonymous, are not good:
        if (type.getEnclosingMethod() != null) {
            return "local/anonymous";
        }

        /* But how about non-static inner classes? Can't construct
         * easily (theoretically, we could try to check if parent
         * happens to be enclosing... but that gets convoluted)
```

```
         */
        if (type.getEnclosingClass() != null) {
            if (!Modifier.isStatic(type.getModifiers())) {
                return "non-static member class";
            }
        }
    }
    catch (SecurityException e) { }
    catch (NullPointerException e) { }
    return null;
}

/**
 * Helper method used to weed out dynamic Proxy types; types that do
 * not expose concrete method API that we could use to figure out
 * automatic Bean (property) based serialization.
 */
public static boolean isProxyType(Class<?> type)
{
    // Then: well-known proxy (etc) classes
    if (Proxy.isProxyClass(type)) {
        return true;
    }
    String name = type.getName();
    // Hibernate uses proxies heavily as well:
    if (name.startsWith("net.sf.cglib.proxy.")
        || name.startsWith("org.hibernate.proxy.")) {
        return true;
    }
    // Not one of known proxies, nope:
    return false;
}

/**
 * Helper method that checks if given class is a concrete one;
 * that is, not an interface or abstract class.
 */
public static boolean isConcrete(Class<?> type)
{
    int mod = type.getModifiers();
    return (mod & (Modifier.INTERFACE | Modifier.ABSTRACT)) == 0;
}

/**
 * @since 1.6
 */
public static boolean isConcrete(Member member)
{
```

```
    int mod = member.getModifiers();
    return (mod & (Modifier.INTERFACE | Modifier.ABSTRACT)) == 0;
}

public static boolean isCollectionMapOrArray(Class<?> type)
{
    if (type.isArray()) return true;
    if (Collection.class.isAssignableFrom(type)) return true;
    if (Map.class.isAssignableFrom(type)) return true;
    return false;
}

/*
/**********************************************************
/* Type name handling methods
/**********************************************************
 */

/**
 * Helper method used to construct appropriate description
 * when passed either type (Class) or an instance; in latter
 * case, class of instance is to be used.
 */
public static String getClassDescription(Object classOrInstance)
{
    if (classOrInstance == null) {
        return "unknown";
    }
    Class<?> cls = (classOrInstance instanceof Class<?>) ?
        (Class<?>) classOrInstance : classOrInstance.getClass();
    return cls.getName();
}

/*
/**********************************************************
/* Method type detection methods
/**********************************************************
 */

public static boolean hasGetterSignature(Method m)
{
    // First: static methods can't be getters
    if (Modifier.isStatic(m.getModifiers())) {
        return false;
    }
    // Must take no args
    Class<?>[] pts = m.getParameterTypes();
    if (pts != null && pts.length != 0) {
```

```
            return false;
        }
        // Can't be a void method
        if (Void.TYPE == m.getReturnType()) {
            return false;
        }
        // Otherwise looks ok:
        return true;
    }

    /*
    /**********************************************************
    /* Exception handling
    /**********************************************************
     */

    /**
     * Method that can be used to find the "root cause", innermost
     * of chained (wrapped) exceptions.
     */
    public static Throwable getRootCause(Throwable t)
    {
        while (t.getCause() != null) {
            t = t.getCause();
        }
        return t;
    }

    /**
     * Method that will unwrap root causes of given Throwable, and throw
     * the innermost {@link Exception} or {@link Error} as is.
     * This is useful in cases where mandatory wrapping is added, which
     * is often done by Reflection API.
     *
     * @since 1.7
     */
    public static void throwRootCause(Throwable t) throws Exception
    {
        t = getRootCause(t);
        if (t instanceof Exception) {
            throw (Exception) t;
        }
        throw (Error) t;
    }

    /**
     * Method that will wrap 't' as an {@link IllegalArgumentException} if it
     * is a checked exception; otherwise (runtime exception or error) throw as is
```

```
 */
public static void throwAsIAE(Throwable t)
{
   throwAsIAE(t, t.getMessage());
}

/**
 * Method that will wrap 't' as an {@link IllegalArgumentException} (and with
 * specified message) if it
 * is a checked exception; otherwise (runtime exception or error) throw as is
 */
public static void throwAsIAE(Throwable t, String msg)
{
   if (t instanceof RuntimeException) {
      throw (RuntimeException) t;
   }
   if (t instanceof Error) {
      throw (Error) t;
   }
   throw new IllegalArgumentException(msg, t);
}

/**
 * Method that will locate the innermost exception for given Throwable;
 * and then wrap it as an {@link IllegalArgumentException} if it
 * is a checked exception; otherwise (runtime exception or error) throw as is
 */
public static void unwrapAndThrowAsIAE(Throwable t)
{
   throwAsIAE(getRootCause(t));
}

/**
 * Method that will locate the innermost exception for given Throwable;
 * and then wrap it as an {@link IllegalArgumentException} if it
 * is a checked exception; otherwise (runtime exception or error) throw as is
 */
public static void unwrapAndThrowAsIAE(Throwable t, String msg)
{
   throwAsIAE(getRootCause(t), msg);
}

/*
/************************************************************
/* Instantiation
/************************************************************
 */
```

```
    /**
     * Method that can be called to try to create an instantiate of
     * specified type. Instantiation is done using default no-argument
     * constructor.
     *
     * @param canFixAccess Whether it is possible to try to change access
     *   rights of the default constructor (in case it is not publicly
     *   accessible) or not.
     *
     * @throws IllegalArgumentException If instantiation fails for any reason;
     *    except for cases where constructor throws an unchecked exception
     *    (which will be passed as is)
     */
    public static <T> T createInstance(Class<T> cls, boolean canFixAccess)
        throws IllegalArgumentException
    {
        Constructor<T> ctor = findConstructor(cls, canFixAccess);
        if (ctor == null) {
            throw new IllegalArgumentException("Class "+cls.getName()+" has no default (no arg) constructor");
        }
        try {
            return ctor.newInstance();
        } catch (Exception e) {
            ClassUtil.unwrapAndThrowAsIAE(e, "Failed to instantiate class "+cls.getName()+", problem:
"+e.getMessage());
            return null;
        }
    }

    public static <T> Constructor<T> findConstructor(Class<T> cls, boolean canFixAccess)
        throws IllegalArgumentException
    {
        try {
            Constructor<T> ctor = cls.getDeclaredConstructor();
            if (canFixAccess) {
                checkAndFixAccess(ctor);
            } else {
                // Has to be public...
                if (!Modifier.isPublic(ctor.getModifiers())) {
                    throw new IllegalArgumentException("Default constructor for "+cls.getName()+" is not accessible
(non-public?): not allowed to try modify access via Reflection: can not instantiate type");
                }
            }
            return ctor;
        } catch (NoSuchMethodException e) {
            ;
        } catch (Exception e) {
            ClassUtil.unwrapAndThrowAsIAE(e, "Failed to find default constructor of class "+cls.getName()+",
```

```
problem: "+e.getMessage());
        }
        return null;
    }

    /*
    /**********************************************************
    /* Primitive type support
    /**********************************************************
     */

    /**
     * Helper method used to get default value for wrappers used for primitive types
     * (0 for Integer etc)
     *
     * @since 1.6.1
     */
    public static Object defaultValue(Class<?> cls)
    {
        if (cls == Integer.TYPE) {
            return Integer.valueOf(0);
        }
        if (cls == Long.TYPE) {
            return Long.valueOf(0L);
        }
        if (cls == Boolean.TYPE) {
            return Boolean.FALSE;
        }
        if (cls == Double.TYPE) {
            return Double.valueOf(0.0);
        }
        if (cls == Float.TYPE) {
            return Float.valueOf(0.0f);
        }
        if (cls == Byte.TYPE) {
            return Byte.valueOf((byte) 0);
        }
        if (cls == Short.TYPE) {
            return Short.valueOf((short) 0);
        }
        if (cls == Character.TYPE) {
            return '\0';
        }
        throw new IllegalArgumentException("Class "+cls.getName()+" is not a primitive type");
    }

    /**
     * Helper method for finding wrapper type for given primitive type (why isn't
```

```
 * there one in JDK?)
 *
 * @since 1.7.1
 */
public static Class<?> wrapperType(Class<?> primitiveType)
{
   if (primitiveType == Integer.TYPE) {
      return Integer.class;
   }
   if (primitiveType == Long.TYPE) {
      return Long.class;
   }
   if (primitiveType == Boolean.TYPE) {
      return Boolean.class;
   }
   if (primitiveType == Double.TYPE) {
      return Double.class;
   }
   if (primitiveType == Float.TYPE) {
      return Float.class;
   }
   if (primitiveType == Byte.TYPE) {
      return Byte.class;
   }
   if (primitiveType == Short.TYPE) {
      return Short.class;
   }
   if (primitiveType == Character.TYPE) {
      return Character.class;
   }
   throw new IllegalArgumentException("Class "+primitiveType.getName()+" is not a primitive type");
}

/*
/**********************************************************
/* Access checking/handling methods
/**********************************************************
 */

/**
 * Method called to check if we can use the passed method or constructor
 * (wrt access restriction -- public methods can be called, others
 * usually not); and if not, if there is a work-around for
 * the problem.
 */
public static void checkAndFixAccess(Member member)
{
   // We know all members are also accessible objects...
```

```
        AccessibleObject ao = (AccessibleObject) member;

        /* 14-Jan-2009, tatu: It seems safe and potentially beneficial to
         *   always to make it accessible (latter because it will force
         *   skipping checks we have no use for...), so let's always call it.
         */
        //if (!ao.isAccessible()) {
        try {
            ao.setAccessible(true);
        } catch (SecurityException se) {
            /* 17-Apr-2009, tatu: Related to [JACKSON-101]: this can fail on
             *    platforms like EJB and Google App Engine); so let's
             *    only fail if we really needed it...
             */
            if (!ao.isAccessible()) {
                Class<?> declClass = member.getDeclaringClass();
                throw new IllegalArgumentException("Can not access "+member+" (from class "+declClass.getName()+";
failed to set access: "+se.getMessage());
            }
        }
        //}
    }

    /*
    /**********************************************************
    /* Enum type detection
    /**********************************************************
     */

    /**
     * Helper method that can be used to dynamically figure out
     * enumeration type of given {@link EnumSet}, without having
     * access to its declaration.
     * Code is needed to work around design flaw in JDK.
     *
     * @since 1.5
     */
    public static Class<? extends Enum<?>> findEnumType(EnumSet<?> s)
    {
        // First things first: if not empty, easy to determine
        if (!s.isEmpty()) {
            return findEnumType(s.iterator().next());
        }
        // Otherwise need to locate using an internal field
        return EnumTypeLocator.instance.enumTypeFor(s);
    }

    /**
```

```
  * Helper method that can be used to dynamically figure out
  * enumeration type of given {@link EnumSet}, without having
  * access to its declaration.
  * Code is needed to work around design flaw in JDK.
  *
  * @since 1.5
  */
 public static Class<? extends Enum<?>> findEnumType(EnumMap<?,?> m)
 {
  if (!m.isEmpty()) {
   return findEnumType(m.keySet().iterator().next());
  }
  // Otherwise need to locate using an internal field
  return EnumTypeLocator.instance.enumTypeFor(m);
 }

 /**
  * Helper method that can be used to dynamically figure out formal
  * enumeration type (class) for given enumeration. This is either
  * class of enum instance (for "simple" enumerations), or its
  * superclass (for enums with instance fields or methods)
  */
 @SuppressWarnings("unchecked")
public static Class<? extends Enum<?>> findEnumType(Enum<?> en)
 {
// enums with "body" are sub-classes of the formal type
  Class<?> ec = en.getClass();
if (ec.getSuperclass() != Enum.class) {
 ec = ec.getSuperclass();
}
return (Class<? extends Enum<?>>) ec;
 }

 /**
  * Helper method that can be used to dynamically figure out formal
  * enumeration type (class) for given class of an enumeration value.
  * This is either class of enum instance (for "simple" enumerations),
  * or its superclass (for enums with instance fields or methods)
  */
 @SuppressWarnings("unchecked")
public static Class<? extends Enum<?>> findEnumType(Class<?> cls)
 {
// enums with "body" are sub-classes of the formal type
if (cls.getSuperclass() != Enum.class) {
 cls = cls.getSuperclass();
}
return (Class<? extends Enum<?>>) cls;
 }
```

```
/*
/**********************************************************
/* Helper classes
/**********************************************************
 */

/**
 * Inner class used to contain gory details of how we can determine
 * details of instances of common JDK types like {@link EnumMap}s.
 */
private static class EnumTypeLocator
{
 final static EnumTypeLocator instance = new EnumTypeLocator();

 private final Field enumSetTypeField;
 private final Field enumMapTypeField;

 private EnumTypeLocator() {
/* JDK uses following fields to store information about actual Enumeration
 * type for EnumSets, EnumMaps...
 */
enumSetTypeField = locateField(EnumSet.class, "elementType", Class.class);
enumMapTypeField = locateField(EnumMap.class, "elementType", Class.class);
 }

 @SuppressWarnings("unchecked")
 public Class<? extends Enum<?>> enumTypeFor(EnumSet<?> set)
 {
  if (enumSetTypeField != null) {
   return (Class<? extends Enum<?>>) get(set, enumSetTypeField);
  }
  throw new IllegalStateException("Can not figure out type for EnumSet (odd JDK platform?)");
}

 @SuppressWarnings("unchecked")
 public Class<? extends Enum<?>> enumTypeFor(EnumMap<?,?> set)
      {
  if (enumMapTypeField != null) {
   return (Class<? extends Enum<?>>) get(set, enumMapTypeField);
  }
  throw new IllegalStateException("Can not figure out type for EnumMap (odd JDK platform?)");

    }

 private Object get(Object bean, Field field)
 {
  try {
```

```
     return field.get(bean);
    } catch (Exception e) {
     throw new IllegalArgumentException(e);
    }
   }

   private static Field locateField(Class<?> fromClass, String expectedName, Class<?> type)
   {
    Field found = null;
    // First: let's see if we can find exact match:
    Field[] fields = fromClass.getDeclaredFields();
    for (Field f : fields) {
     if (expectedName.equals(f.getName()) && f.getType() == type) {
      found = f;
      break;
     }
    }
    // And if not, if there is just one field with the type, that field
    if (found == null) {
      for (Field f : fields) {
       if (f.getType() == type) {
        // If more than one, can't choose
        if (found != null) return null;
        found = f;
       }
      }
    }
    if (found != null) { // it's non-public, need to force accessible
     try {
      found.setAccessible(true);
     } catch (Throwable t) { }
    }
    return found;
   }
  }
}
package org.codehaus.jackson.map.util;

import java.lang.annotation.Annotation;

/**
* Interface that defines interface for collection of annotations.
*<p>
* Standard mutable implementation is {@link org.codehaus.jackson.map.introspect.AnnotationMap}
*
* @since 1.7
*/
public interface Annotations
```

```
{
  /**
   * Main access method used to find value for given annotation.
   */
  public <A extends Annotation> A get(Class<A> cls);

  /**
   * Returns number of annotation entries in this collection.
   */
  public int size();
}
package org.codehaus.jackson.map;

import java.io.*;
import java.net.URL;

import org.codehaus.jackson.*;
import org.codehaus.jackson.node.JsonNodeFactory;
import org.codehaus.jackson.node.NullNode;

/**
* This mapper (or, codec) provides mapping between Json,
* and Tree-like structure that consists of child-linked
* nodes that can be traversed with simple path operations
* (indexing arrays by element, objects by field name).
*<p>
* As of version 0.9.9 of Jackson, most functionality has
* been moved away from this class: all serialization and
* deserialization features are now accessible through either
* {@link ObjectMapper} or {@link JsonParser} and
* {@link JsonGenerator}. The remaining functionality is limited
* to {@link JsonNodeFactory} implementation which allows constructing
* typed {@link JsonNode} instances.
*
* @deprecated Use {@link org.codehaus.jackson.map.ObjectMapper} instead
*/
@Deprecated
public class TreeMapper
  extends JsonNodeFactory
{
  /**
   * Mapper that handles actual serialization/deserialization
   */
  protected ObjectMapper _objectMapper;

  /*
  /**********************************************************
  /* Life-cycle (construction, configuration)
```

```
/**********************************************************
 */

public TreeMapper()
{
   this(null);
}

public TreeMapper(ObjectMapper m)
{
   _objectMapper = m;
}

/**
 * Method that can be used to get hold of Json factory that this
 * mapper uses if it needs to construct Json parsers and/or generators.
 *
 * @return Json factory that this mapper uses when it needs to
 *   construct Json parser and generators
 */
public JsonFactory getJsonFactory() { return objectMapper().getJsonFactory(); }

/*
/**********************************************************
/* Public API, constructing in-memory trees
/**********************************************************
 */

/*
/**********************************************************
/* Public API, root-level mapping methods,
/* mapping from JSON content to nodes
/**********************************************************
 */

/**
 * Method that will try to read a sub-tree using given parser,
 * map it to a tree (represented by a root JsonNode) and return
 * it, if possible. Alternatively, if no content is available,
 * null is returned to signal end-of-content.
 */
public JsonNode readTree(JsonParser jp)
   throws IOException, JsonParseException
{
   /* 02-Mar-2009, tatu: Behavior here is bit different from that
    *   of ObjectMapper, since we can actually return null to
    *   indicate end-of-content. Because of this we do need to
    *   check for EOF here and not let ObjectMapper encounter
```

```
     *   it (since that would throw an exception)
     */
    JsonToken t = jp.getCurrentToken();
    if (t == null) {
        t = jp.nextToken();
        if (t == null) {
            return null;
        }
    }
    // note: called method converts null to NullNode:
    return objectMapper().readTree(jp);
}

public JsonNode readTree(File src)
    throws IOException, JsonParseException
{

    JsonNode n = objectMapper().readValue(src, JsonNode.class);
    return (n == null) ? NullNode.instance : n;
}

public JsonNode readTree(URL src)
    throws IOException, JsonParseException
{
    JsonNode n = objectMapper().readValue(src, JsonNode.class);
    return (n == null) ? NullNode.instance : n;
}

public JsonNode readTree(InputStream src)
    throws IOException, JsonParseException
{
    JsonNode n = objectMapper().readValue(src, JsonNode.class);
    return (n == null) ? NullNode.instance : n;
}

public JsonNode readTree(Reader src)
    throws IOException, JsonParseException
{
    JsonNode n = objectMapper().readValue(src, JsonNode.class);
    return (n == null) ? NullNode.instance : n;
}

public JsonNode readTree(String jsonContent)
    throws IOException, JsonParseException
{
    JsonNode n = objectMapper().readValue(jsonContent, JsonNode.class);
    return (n == null) ? NullNode.instance : n;
}
```

```
  public JsonNode readTree(byte[] jsonContent)
     throws IOException, JsonParseException
  {
     JsonNode n = objectMapper().readValue(jsonContent, 0, jsonContent.length, JsonNode.class);
     return (n == null) ? NullNode.instance : n;
  }

  /*
  /**********************************************************
  /* Public API, root-level mapping methods,
  /* writing nodes as JSON content
  /**********************************************************
   */

  public void writeTree(JsonNode rootNode, File dst)
     throws IOException, JsonParseException
  {
     objectMapper().writeValue(dst, rootNode);
  }

  public void writeTree(JsonNode rootNode, Writer dst)
     throws IOException, JsonParseException
  {
     objectMapper().writeValue(dst, rootNode);
  }

  public void writeTree(JsonNode rootNode, OutputStream dst)
     throws IOException, JsonParseException
  {
     objectMapper().writeValue(dst, rootNode);
  }

  /*
  /**********************************************************
  /* Internal methods
  /**********************************************************
   */

  protected synchronized ObjectMapper objectMapper()
  {
     if (_objectMapper == null) {
        _objectMapper = new ObjectMapper();
     }
     return _objectMapper;
  }
}
```

```
package org.codehaus.jackson.map;

import java.io.*;
import java.net.URL;
import java.util.concurrent.ConcurrentHashMap;

import org.codehaus.jackson.*;
import org.codehaus.jackson.map.deser.StdDeserializationContext;
import org.codehaus.jackson.map.introspect.VisibilityChecker;
import org.codehaus.jackson.map.jsontype.SubtypeResolver;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.node.JsonNodeFactory;
import org.codehaus.jackson.node.NullNode;
import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.util.VersionUtil;

/**
* Builder object that can be used for per-serialization configuration of
* deserialization parameters, such as root type to use or object
* to update (instead of constructing new instance).
* Uses "fluid" (aka builder) pattern so that instances are immutable
* (and thus fully thread-safe with no external synchronization);
* new instances are constructed for different configurations.
* Instances are initially constructed by {@link ObjectMapper} and can be
* reused.
*
* @author tatu
* @since 1.6
*/
public class ObjectReader
  implements Versioned
{
  private final static JavaType JSON_NODE_TYPE = TypeFactory.type(JsonNode.class);

  /*
  /**********************************************************
  /* Immutable configuration from ObjectMapper
  /**********************************************************
   */

  /**
   * Root-level cached deserializers
   */
  final protected ConcurrentHashMap<JavaType, JsonDeserializer<Object>> _rootDeserializers;

  /**
   * General serialization configuration settings
```

```
 */
protected final DeserializationConfig _config;

protected final DeserializerProvider _provider;

/**
 * Factory used for constructing {@link JsonGenerator}s
 */
protected final JsonFactory _jsonFactory;

// Support for polymorphic types:
protected TypeResolverBuilder<?> _defaultTyper;

// Configurable visibility limits
protected VisibilityChecker<?> _visibilityChecker;

/**
 * Registered concrete subtypes that can be used instead of (or
 * in addition to) ones declared using annotations.
 *
 * @since 1.6
 */
protected final SubtypeResolver _subtypeResolver;

/*
/**********************************************************
/* Configuration that can be changed during building
/**********************************************************
 */

/**
 * Declared type of value to instantiate during deserialization.
 * Defines which deserializer to use; as well as base type of instance
 * to construct if an updatable value is not configured to be used
 * (subject to changes by embedded type information, for polymorphic
 * types). If {@link #_valueToUpdate} is non-null, only used for
 * locating deserializer.
 */
protected final JavaType _valueType;

/**
 * Instance to update with data binding; if any. If null,
 * a new instance is created, if non-null, properties of
 * this value object will be updated instead.
 * Note that value can be of almost any type, except not
 * {@link org.codehaus.jackson.map.type.ArrayType}; array
 * types can not be modified because array size is immutable.
 */
```

```
protected final Object _valueToUpdate;

/*
/**********************************************************
/* Life-cycle
/**********************************************************
 */

/**
 * Constructor used by {@link ObjectMapper} for initial instantiation
 */
protected ObjectReader(ObjectMapper mapper,  JavaType valueType, Object valueToUpdate)
{
    _rootDeserializers = mapper._rootDeserializers;
    _defaultTyper = mapper._defaultTyper;
    _visibilityChecker = mapper._visibilityChecker;
    _subtypeResolver = mapper._subtypeResolver;
    _provider = mapper._deserializerProvider;
    _jsonFactory = mapper._jsonFactory;

    // must make a copy at this point, to prevent further changes from trickling down
    _config = mapper._deserializationConfig.createUnshared(_defaultTyper, _visibilityChecker,
            _subtypeResolver);

    _valueType = valueType;
    _valueToUpdate = valueToUpdate;
    if (valueToUpdate != null && valueType.isArrayType()) {
        throw new IllegalArgumentException("Can not update an array value");
    }
}

/**
 * Copy constructor used for building variations.
 */
protected ObjectReader(ObjectReader base, DeserializationConfig config,
        JavaType valueType, Object valueToUpdate)
{
    _rootDeserializers = base._rootDeserializers;
    _defaultTyper = base._defaultTyper;
    _visibilityChecker = base._visibilityChecker;
    _provider = base._provider;
    _jsonFactory = base._jsonFactory;
    _subtypeResolver = base._subtypeResolver;

    _config = config;

    _valueType = valueType;
    _valueToUpdate = valueToUpdate;
```

```
    if (valueToUpdate != null && valueType.isArrayType()) {
        throw new IllegalArgumentException("Can not update an array value");
    }
}

/**
 * Method that will return version information stored in and read from jar
 * that contains this class.
 *
 * @since 1.6
 */
public Version version() {
    return VersionUtil.versionFor(getClass());
}

public ObjectReader withType(JavaType valueType)
{
    if (valueType == _valueType) return this;
    // type is stored here, no need to make a copy of config
    return new ObjectReader(this, _config, valueType, _valueToUpdate);
}

public ObjectReader withType(Class<?> valueType)
{
    return withType(TypeFactory.type(valueType));
}

public ObjectReader withType(java.lang.reflect.Type valueType)
{
    return withType(TypeFactory.type(valueType));
}

public ObjectReader withNodeFactory(JsonNodeFactory f)
{
    // node factory is stored within config, so need to copy that first
    if (f == _config.getNodeFactory()) return this;
    DeserializationConfig cfg = _config.createUnshared(f);
    return new ObjectReader(this, cfg, _valueType, _valueToUpdate);
}

public ObjectReader withValueToUpdate(Object value)
{
    if (value == _valueToUpdate) return this;
    if (value == null) {
        throw new IllegalArgumentException("cat not update null value");
    }
    JavaType t = TypeFactory.type(value.getClass());
    return new ObjectReader(this, _config, t, value);
```

```
}

/*
/**********************************************************
/* Deserialization methods; basic ones to support ObjectCodec first
/* (ones that take JsonParser)
/**********************************************************
 */

@SuppressWarnings("unchecked")
public <T> T readValue(JsonParser jp)
   throws IOException, JsonProcessingException
{
   return (T) _bind(jp);
}

public JsonNode readTree(JsonParser jp)
   throws IOException, JsonProcessingException
{
   return _bindAsTree(jp);
}

/*
/**********************************************************
/* Deserialization methods; others similar to what ObjectMapper has
/**********************************************************
 */

@SuppressWarnings("unchecked")
public <T> T readValue(InputStream src)
   throws IOException, JsonProcessingException
{
   return (T) _bindAndClose(_jsonFactory.createJsonParser(src));
}

@SuppressWarnings("unchecked")
public <T> T readValue(Reader src)
   throws IOException, JsonProcessingException
{
   return (T) _bindAndClose(_jsonFactory.createJsonParser(src));
}

@SuppressWarnings("unchecked")
public <T> T readValue(String src)
   throws IOException, JsonProcessingException
{
   return (T) _bindAndClose(_jsonFactory.createJsonParser(src));
}
```

```
@SuppressWarnings("unchecked")
public <T> T readValue(byte[] src)
   throws IOException, JsonProcessingException
{
   return (T) _bindAndClose(_jsonFactory.createJsonParser(src));
}

@SuppressWarnings("unchecked")
public <T> T readValue(byte[] src, int offset, int length)
   throws IOException, JsonProcessingException
{
   return (T) _bindAndClose(_jsonFactory.createJsonParser(src, offset, length));
}

@SuppressWarnings("unchecked")
public <T> T readValue(File src)
   throws IOException, JsonProcessingException
{
   return (T) _bindAndClose(_jsonFactory.createJsonParser(src));
}

@SuppressWarnings("unchecked")
public <T> T readValue(URL src)
   throws IOException, JsonProcessingException
{
   return (T) _bindAndClose(_jsonFactory.createJsonParser(src));
}

/**
 * Convenience method for converting results from given JSON tree into given
 * value type. Basically short-cut for:
 *<pre>
 *   objectReader.readValue(src.traverse())
 *</pre>
 *
 * @since 1.6
 */
@SuppressWarnings("unchecked")
public <T> T readValue(JsonNode src)
   throws IOException, JsonProcessingException
{
   return (T) _bindAndClose(src.traverse());
}

public JsonNode readTree(InputStream in)
   throws IOException, JsonProcessingException
{
```

```
    return _bindAndCloseAsTree(_jsonFactory.createJsonParser(in));
}

public JsonNode readTree(Reader r)
    throws IOException, JsonProcessingException
{
    return _bindAndCloseAsTree(_jsonFactory.createJsonParser(r));
}

public JsonNode readTree(String content)
    throws IOException, JsonProcessingException
{
    return _bindAndCloseAsTree(_jsonFactory.createJsonParser(content));
}

/*
/**********************************************************
/* Helper methods
/**********************************************************
 */

/**
 * Actual implementation of value reading+binding operation.
 */
protected Object _bind(JsonParser jp)
    throws IOException, JsonParseException, JsonMappingException
{
    /* First: may need to read the next token, to initialize state (either
     * before first read from parser, or after previous token has been cleared)
     */
    Object result;
    JsonToken t = _initForReading(jp);
    if (t == JsonToken.VALUE_NULL || t == JsonToken.END_ARRAY || t == JsonToken.END_OBJECT) {
        result = _valueToUpdate;
    } else { // pointing to event other than null
        DeserializationContext ctxt = _createDeserializationContext(jp, _config);
        if (_valueToUpdate == null) {
            result = _findRootDeserializer(_config, _valueType).deserialize(jp, ctxt);
        } else {
            _findRootDeserializer(_config, _valueType).deserialize(jp, ctxt, _valueToUpdate);
            result = _valueToUpdate;
        }
    }
    // Need to consume the token too
    jp.clearCurrentToken();
    return result;
}
```

```
protected Object _bindAndClose(JsonParser jp)
   throws IOException, JsonParseException, JsonMappingException
{
   try {
      Object result;
      JsonToken t = _initForReading(jp);
      if (t == JsonToken.VALUE_NULL || t == JsonToken.END_ARRAY || t == JsonToken.END_OBJECT) {
         result = _valueToUpdate;
      } else {
         DeserializationContext ctxt = _createDeserializationContext(jp, _config);
         if (_valueToUpdate == null) {
            result = _findRootDeserializer(_config, _valueType).deserialize(jp, ctxt);
         } else {
            _findRootDeserializer(_config, _valueType).deserialize(jp, ctxt, _valueToUpdate);
            result = _valueToUpdate;
         }
      }
      return result;
   } finally {
      try {
         jp.close();
      } catch (IOException ioe) { }
   }
}

protected JsonNode _bindAsTree(JsonParser jp)
   throws IOException, JsonParseException, JsonMappingException
{
   JsonNode result;
   JsonToken t = _initForReading(jp);
   if (t == JsonToken.VALUE_NULL || t == JsonToken.END_ARRAY || t == JsonToken.END_OBJECT) {
      result = NullNode.instance;
   } else {
      DeserializationContext ctxt = _createDeserializationContext(jp, _config);
      // Bit more complicated, since we may need to override node factory:
      result = (JsonNode) _findRootDeserializer(_config, JSON_NODE_TYPE).deserialize(jp, ctxt);
   }
   // Need to consume the token too
   jp.clearCurrentToken();
   return result;
}

protected JsonNode _bindAndCloseAsTree(JsonParser jp)
   throws IOException, JsonParseException, JsonMappingException
{
   try {
      return _bindAsTree(jp);
   } finally {
```

```
        try {
            jp.close();
        } catch (IOException ioe) { }
    }
}

protected static JsonToken _initForReading(JsonParser jp)
    throws IOException, JsonParseException, JsonMappingException
{
    /* First: must point to a token; if not pointing to one, advance.
     * This occurs before first read from JsonParser, as well as
     * after clearing of current token.
     */
    JsonToken t = jp.getCurrentToken();
    if (t == null) { // and then we must get something...
        t = jp.nextToken();
        if (t == null) { // [JACKSON-99] Should throw EOFException?
            throw new EOFException("No content to map to Object due to end of input");
        }
    }
    return t;
}

/**
 * Method called to locate deserializer for the passed root-level value.
 */
protected JsonDeserializer<Object> _findRootDeserializer(DeserializationConfig cfg, JavaType valueType)
    throws JsonMappingException
{
    // First: have we already seen it?
    JsonDeserializer<Object> deser = _rootDeserializers.get(valueType);
    if (deser != null) {
        return deser;
    }

    // Nope: need to ask provider to resolve it
    deser = _provider.findTypedValueDeserializer(cfg, valueType, null);
    if (deser == null) { // can this happen?
        throw new JsonMappingException("Can not find a deserializer for type "+valueType);
    }
    _rootDeserializers.put(valueType, deser);
    return deser;
}

protected DeserializationContext _createDeserializationContext(JsonParser jp, DeserializationConfig cfg) {
    // 04-Jan-2010, tatu: we do actually need the provider too... (for polymorphic deser)
    return new StdDeserializationContext(cfg, jp, _provider);
}
```

```
}
/**
* Package that contains interfaces that define how to implement
* functionality for dynamically resolving type during deserialization.
* This is needed for complete handling of polymorphic types, where
* actual type can not be determined statically (declared type is
* a supertype of actual polymorphic serialized types).
*
* @since 1.5
*/
package org.codehaus.jackson.map.jsontype;
/**
* Package that contains standard implementations for
* {@link org.codehaus.jackson.map.jsontype.TypeResolverBuilder}
* and
* {@link org.codehaus.jackson.map.jsontype.TypeIdResolver}.
*
* @since 1.5
*/
package org.codehaus.jackson.map.jsontype.impl;
package org.codehaus.jackson.map.jsontype.impl;

import java.io.IOException;

import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.annotate.JsonTypeInfo.As;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;

/**
* Type wrapper that tries to use an extra JSON Object, with a single
* entry that has type name as key, to serialize type information.
* If this is not possible (value is serialize as array or primitive),
* will use {@link As#WRAPPER_ARRAY} mechanism as fallback: that is,
* just use a wrapping array with type information as the first element
* and value as second.
*
* @since 1.5
* @author tatus
*/
public class AsWrapperTypeSerializer
  extends TypeSerializerBase
{
  public AsWrapperTypeSerializer(TypeIdResolver idRes, BeanProperty property)
  {
     super(idRes, property);
  }
```

```
@Override
public As getTypeInclusion() { return As.WRAPPER_OBJECT; }

@Override
public void writeTypePrefixForObject(Object value, JsonGenerator jgen)
   throws IOException, JsonProcessingException
{
   // wrapper
   jgen.writeStartObject();
   // and then JSON Object start caller wants
   jgen.writeObjectFieldStart(_idResolver.idFromValue(value));
}

@Override
public void writeTypePrefixForArray(Object value, JsonGenerator jgen)
   throws IOException, JsonProcessingException
{
   // can still wrap ok
   jgen.writeStartObject();
   // and then JSON Array start caller wants
   jgen.writeArrayFieldStart(_idResolver.idFromValue(value));
}

@Override
public void writeTypePrefixForScalar(Object value, JsonGenerator jgen)
      throws IOException, JsonProcessingException
{
   // can still wrap ok
   jgen.writeStartObject();
   jgen.writeFieldName(_idResolver.idFromValue(value));
}

@Override
public void writeTypeSuffixForObject(Object value, JsonGenerator jgen)
   throws IOException, JsonProcessingException
{
   // first close JSON Object caller used
   jgen.writeEndObject();
   // and then wrapper
   jgen.writeEndObject();
}

@Override
public void writeTypeSuffixForArray(Object value, JsonGenerator jgen)
   throws IOException, JsonProcessingException
{
   // first close array caller needed
```

```
        jgen.writeEndArray();
        // then wrapper object
        jgen.writeEndObject();
    }

    @Override
    public void writeTypeSuffixForScalar(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException
    {
        // just need to close the wrapper object
        jgen.writeEndObject();
    }
}
package org.codehaus.jackson.map.jsontype.impl;

import java.util.*;

import org.codehaus.jackson.map.AnnotationIntrospector;
import org.codehaus.jackson.map.MapperConfig;
import org.codehaus.jackson.map.introspect.*;
import org.codehaus.jackson.map.jsontype.NamedType;
import org.codehaus.jackson.map.jsontype.SubtypeResolver;

public class StdSubtypeResolver extends SubtypeResolver
{
    protected LinkedHashSet<NamedType> _registeredSubtypes;

    public StdSubtypeResolver() { }

    /*
    /**********************************************************
    /* Public API
    /**********************************************************
     */

    @Override
    public void registerSubtypes(NamedType... types)
    {
        if (_registeredSubtypes == null) {
            _registeredSubtypes = new LinkedHashSet<NamedType>();
        }
        for (NamedType type : types) {
            _registeredSubtypes.add(type);
        }
    }

    @Override
    public void registerSubtypes(Class<?>... classes)
```

```
{
    NamedType[] types = new NamedType[classes.length];
    for (int i = 0, len = classes.length; i < len; ++i) {
        types[i] = new NamedType(classes[i]);
    }
    registerSubtypes(types);
}

/**
 *
 * @param property Base member to use for type resolution: either annotated type (class),
 *    or property (field, getter/setter)
 */
@Override
public Collection<NamedType> collectAndResolveSubtypes(AnnotatedMember property,
    MapperConfig<?> config, AnnotationIntrospector ai)
{
    // but if annotations found, may need to resolve subtypes:
    Collection<NamedType> st = ai.findSubtypes(property);
    // If no explicit definitions, base itself might have name
    if (st == null) {
        st = new ArrayList<NamedType>();
    }
    // ensure that base type is included as starting point
    st.add(new NamedType(property.getRawType(), null));
    return _collectAndResolve(property, config, ai, st);
}

@Override
public Collection<NamedType> collectAndResolveSubtypes(AnnotatedClass type,
        MapperConfig<?> config, AnnotationIntrospector ai)
{
    HashMap<NamedType, NamedType> subtypes = new HashMap<NamedType, NamedType>();
    // [JACKSON-257] then consider registered subtypes (which have precedence over annotations)
    if (_registeredSubtypes != null) {
        Class<?> rawBase = type.getRawType();
        for (NamedType subtype : _registeredSubtypes) {
            // is it a subtype of root type?
            if (rawBase.isAssignableFrom(subtype.getType())) { // yes
                AnnotatedClass curr = AnnotatedClass.constructWithoutSuperTypes(subtype.getType(), ai, config);
                _collectAndResolve(curr, subtype, config, ai, subtypes);
            }
        }
    }
    // and then check subtypes via annotations from base type (recursively)
    NamedType rootType = new NamedType(type.getRawType(), null);
    _collectAndResolve(type, rootType, config, ai, subtypes);
    return new ArrayList<NamedType>(subtypes.values());
```

```
}

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

/**
 * Method called to find subtypes for a property that has specific annotations
 * (mostly to support JAXB per-property subtype declarations)
 *
 * @param property Member (method, field) to proces
 */
protected Collection<NamedType> _collectAndResolve(AnnotatedMember property, MapperConfig<?> config,
        AnnotationIntrospector ai, Collection<NamedType> subtypeList)
{
    // Hmmh. Can't iterate over collection and modify it, so:
    HashSet<NamedType> seen = new HashSet<NamedType>(subtypeList);
    ArrayList<NamedType> subtypes = new ArrayList<NamedType>(subtypeList);

    // collect all subtypes iteratively
    for (int i = 0; i < subtypes.size(); ++i) {
        NamedType type = subtypes.get(i);
        AnnotatedClass ac = AnnotatedClass.constructWithoutSuperTypes(type.getType(), ai, config);
        // but first: does type have a name already?
        if (!type.hasName()) { // if not, let's see if annotations define it
            type.setName(ai.findTypeName(ac));
        }
        // and see if annotations list more subtypes
        List<NamedType> moreTypes = ai.findSubtypes(ac);
        if (moreTypes != null) {
            for (NamedType t2 : moreTypes) {
                // we want to keep the first reference (may have name)
                if (seen.add(t2)) {
                    subtypes.add(t2);
                }
            }
        }
    }
    // 14-Aug-2010, tatu: Should we consider registered subtypes [JACKSON-257]?
    //   For now assume not, such that annotation is assumed to be complete
    return subtypes;
}

/**
 * Method called to find subtypes for a specific type (class)
 */
```

```
  protected void _collectAndResolve(AnnotatedClass annotatedType, NamedType namedType,
        MapperConfig<?> config, AnnotationIntrospector ai, HashMap<NamedType, NamedType>
collectedSubtypes)
  {
    if (!namedType.hasName()) {
      String name = ai.findTypeName(annotatedType);
      if (name != null) {
        namedType = new NamedType(namedType.getType(), name);
      }
    }

    // First things first: is base type itself included?
    if (collectedSubtypes.containsKey(namedType)) {
      // if so, no recursion; however, may need to update name?
      if (namedType.hasName()) {
        NamedType prev = collectedSubtypes.get(namedType);
        if (!prev.hasName()) {
          collectedSubtypes.put(namedType, namedType);
        }
      }
      return;
    }
    // if it wasn't, add and check subtypes recursively
    collectedSubtypes.put(namedType, namedType);
    Collection<NamedType> st = ai.findSubtypes(annotatedType);
    if (st != null && !st.isEmpty()) {
      for (NamedType subtype : st) {
        AnnotatedClass subtypeClass = AnnotatedClass.constructWithoutSuperTypes(subtype.getType(), ai,
config);
        // One more thing: name may be either in reference, or in subtype:
        if (!subtype.hasName()) {
          subtype = new NamedType(subtype.getType(), ai.findTypeName(subtypeClass));
        }
        _collectAndResolve(subtypeClass, subtype, config, ai, collectedSubtypes);
      }
    }
  }

  /*
  protected Collection<NamedType> _collectAndResolve(AnnotatedClass rootType,
        MapperConfig<?> config, AnnotationIntrospector ai, Collection<NamedType> subtypeList)
  {
    // Hmmh. Can't iterate over collection and modify it, need to make a copy
    HashMap<NamedType,NamedType> subtypes = new HashMap<NamedType,NamedType>(4 + 2 *
subtypeList.size());
    for (NamedType type : subtypeList) {
      subtypes.put(type, type);
    }
```

```
        // Plus root type can have name of its own...
        NamedType rootNamedType = new NamedType(rootType.getRawType(), ai.findTypeName(rootType));

        subtypes.put(rootNamedType, rootNamedType);

        // Check subtypes reachable via annotations first:
        for (int i = 0; i < subtypes.size(); ++i) {
            NamedType type = subtypes.get(i);
            AnnotatedClass ac = AnnotatedClass.constructWithoutSuperTypes(type.getType(), ai, config);
            // but first: does type have a name already?
            if (!type.hasName()) { // if not, let's see if annotations define it
                type.setName(ai.findTypeName(ac));
            }
            // and see if annotations list more subtypes
            List<NamedType> moreTypes = ai.findSubtypes(ac);
            if (moreTypes != null) {
                for (NamedType t2 : moreTypes) {
                    // we want to keep the first reference (may have name)
                    if (seen.add(t2)) {
                        subtypes.add(t2);
                    }
                }
            }
        }

        return subtypes.values();
    }
    */
}
package org.codehaus.jackson.map.jsontype.impl;

import java.io.IOException;

import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.annotate.JsonTypeInfo.As;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;

/**
* Type serializer that will embed type information in an array,
* as the first element, and actual value as the second element.
*
* @since 1.5
* @author tatu
*/
public class AsArrayTypeSerializer
```

```
  extends TypeSerializerBase
{
  public AsArrayTypeSerializer(TypeIdResolver idRes, BeanProperty property)
  {
     super(idRes, property);
  }

  @Override
  public As getTypeInclusion() { return As.WRAPPER_ARRAY; }

  @Override
  public void writeTypePrefixForObject(Object value, JsonGenerator jgen)
     throws IOException, JsonProcessingException
  {
     jgen.writeStartArray();
     jgen.writeString(_idResolver.idFromValue(value));
     jgen.writeStartObject();
  }

  @Override
  public void writeTypePrefixForArray(Object value, JsonGenerator jgen)
     throws IOException, JsonProcessingException
  {
     jgen.writeStartArray();
     jgen.writeString(_idResolver.idFromValue(value));
     jgen.writeStartArray();
  }

  @Override
  public void writeTypePrefixForScalar(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException
  {
     // only need the wrapper array
     jgen.writeStartArray();
     jgen.writeString(_idResolver.idFromValue(value));
  }

  @Override
  public void writeTypeSuffixForObject(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException
  {
     jgen.writeEndObject();
     jgen.writeEndArray();
  }

  @Override
  public void writeTypeSuffixForArray(Object value, JsonGenerator jgen)
        throws IOException, JsonProcessingException
```

```
    {
        // wrapper array first, and then array caller needs to close
        jgen.writeEndArray();
        jgen.writeEndArray();
    }

    @Override
    public void writeTypeSuffixForScalar(Object value, JsonGenerator jgen)
            throws IOException, JsonProcessingException
    {
        // just the wrapper array to close
        jgen.writeEndArray();
    }
}
package org.codehaus.jackson.map.jsontype.impl;

import org.codehaus.jackson.annotate.JsonTypeInfo;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.TypeSerializer;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;

/**
* @since 1.5
* @author tatus
*/
public abstract class TypeSerializerBase extends TypeSerializer
{
    protected final TypeIdResolver _idResolver;

    protected final BeanProperty _property;

    protected TypeSerializerBase(TypeIdResolver idRes, BeanProperty property)
    {
        _idResolver = idRes;
        _property = property;
    }

    @Override
    public abstract JsonTypeInfo.As getTypeInclusion();

    @Override
    public String getPropertyName() { return null; }

    @Override
    public TypeIdResolver getTypeIdResolver() { return _idResolver; }
}
package org.codehaus.jackson.map.jsontype.impl;
```

```
import java.io.IOException;
import java.util.HashMap;

import org.codehaus.jackson.*;
import org.codehaus.jackson.annotate.JsonTypeInfo;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.DeserializationContext;
import org.codehaus.jackson.map.JsonDeserializer;
import org.codehaus.jackson.map.TypeDeserializer;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;
import org.codehaus.jackson.type.JavaType;

/**
* @since 1.5
* @author tatus
*/
public abstract class TypeDeserializerBase extends TypeDeserializer
{
  protected final TypeIdResolver _idResolver;

  protected final JavaType _baseType;

  protected final BeanProperty _property;

  /**
   * For efficient operation we will lazily build mappings from type ids
   * to actual deserializers, once needed.
   */
  protected final HashMap<String,JsonDeserializer<Object>> _deserializers;

  protected TypeDeserializerBase(JavaType baseType, TypeIdResolver idRes, BeanProperty property)
  {
     _baseType = baseType;
     _idResolver = idRes;
     _property = property;
     _deserializers = new HashMap<String,JsonDeserializer<Object>>();
  }

  @Override
  public abstract JsonTypeInfo.As getTypeInclusion();

  public String baseTypeName() { return _baseType.getRawClass().getName(); }

  @Override
  public String getPropertyName() { return null; }

  @Override
  public TypeIdResolver getTypeIdResolver() { return _idResolver; }
```

```
    @Override
    public String toString()
    {
     StringBuilder sb = new StringBuilder();
     sb.append('[').append(getClass().getName());
     sb.append("; base-type:").append(_baseType);
     sb.append("; id-resolver: ").append(_idResolver);
     sb.append(']');
     return sb.toString();
    }

    /*
     ********************************************************************
     * Helper methods for sub-classes
     ********************************************************************
     */

    protected final JsonDeserializer<Object> _findDeserializer(DeserializationContext ctxt, String typeId)
       throws IOException, JsonProcessingException
    {
       JsonDeserializer<Object> deser;

       synchronized (_deserializers) {
          deser = _deserializers.get(typeId);
          if (deser == null) {
             JavaType type = _idResolver.typeFromId(typeId);
             if (type == null) {
                throw ctxt.unknownTypeException(_baseType, typeId);
             }
             /* 16-Dec-2010, tatu: Since nominal type we get here has no (generic) type parameters,
              *   we actually now need to explicitly narrow from base type (which may have parameterization)
              *   using raw type.
              *
              *   One complication, though; can not change 'type class' (simple type to container); otherwise
              *   we may try to narrow a SimpleType (Object.class) into MapType (Map.class), losing actual
              *   type in process (getting SimpleType of Map.class which will not work as expected)
              */
             if (_baseType != null &&  _baseType.getClass() == type.getClass()) {
                type = _baseType.narrowBy(type.getRawClass());
             }
             deser = ctxt.getDeserializerProvider().findValueDeserializer(ctxt.getConfig(), type, _property);
             _deserializers.put(typeId, deser);
          }
       }
       return deser;
    }
}
```

```
package org.codehaus.jackson.map.jsontype.impl;

import java.util.Collection;

import org.codehaus.jackson.annotate.JsonTypeInfo;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.TypeDeserializer;
import org.codehaus.jackson.map.TypeSerializer;
import org.codehaus.jackson.map.jsontype.NamedType;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;
import org.codehaus.jackson.map.jsontype.TypeResolverBuilder;
import org.codehaus.jackson.type.JavaType;

/**
* Default {@link TypeResolverBuilder} implementation.
*
* @author tatu
* @since 1.5
*/
public class StdTypeResolverBuilder
  implements TypeResolverBuilder<StdTypeResolverBuilder>
{
  // Configuration settings:

  protected JsonTypeInfo.Id _idType;

  protected JsonTypeInfo.As _includeAs;

  protected String _typeProperty;

  // Objects

  protected TypeIdResolver _customIdResolver;

  /*
  /**********************************************************
  /* Construction, initialization, actual building
  /**********************************************************
   */

  public StdTypeResolverBuilder() { }

  public StdTypeResolverBuilder init(JsonTypeInfo.Id idType, TypeIdResolver idRes)
  {
     // sanity checks
     if (idType == null) {
        throw new IllegalArgumentException("idType can not be null");
     }
```

```
        _idType = idType;
        _customIdResolver = idRes;
        // Let's also initialize property name as per idType default
        _typeProperty = idType.getDefaultPropertyName();
        return this;
    }

    @Override
    public TypeSerializer buildTypeSerializer(JavaType baseType, Collection<NamedType> subtypes,
            BeanProperty property)
    {
        TypeIdResolver idRes = idResolver(baseType, subtypes, true, false);
        switch (_includeAs) {
        case WRAPPER_ARRAY:
            return new AsArrayTypeSerializer(idRes, property);
        case PROPERTY:
            return new AsPropertyTypeSerializer(idRes, property, _typeProperty);
        case WRAPPER_OBJECT:
            return new AsWrapperTypeSerializer(idRes, property);
        }
        throw new IllegalStateException("Do not know how to construct standard type serializer for inclusion type:
"+_includeAs);
    }

    public TypeDeserializer buildTypeDeserializer(JavaType baseType, Collection<NamedType> subtypes,
            BeanProperty property)
    {
        TypeIdResolver idRes = idResolver(baseType, subtypes, false, true);

        // First, method for converting type info to type id:
        switch (_includeAs) {
        case WRAPPER_ARRAY:
            return new AsArrayTypeDeserializer(baseType, idRes, property);
        case PROPERTY:
            return new AsPropertyTypeDeserializer(baseType, idRes, property, _typeProperty);
        case WRAPPER_OBJECT:
            return new AsWrapperTypeDeserializer(baseType, idRes, property);
        }
        throw new IllegalStateException("Do not know how to construct standard type serializer for inclusion type:
"+_includeAs);
    }

    /*
    /**********************************************************
    /* Construction, configuration
    /**********************************************************
     */
```

```
public StdTypeResolverBuilder inclusion(JsonTypeInfo.As includeAs) {
    if (includeAs == null) {
        throw new IllegalArgumentException("includeAs can not be null");
    }
    _includeAs = includeAs;
    return this;
}

/**
 * Method for constructing an instance with specified type property name
 * (property name to use for type id when using "as-property" inclusion).
 */
public StdTypeResolverBuilder typeProperty(String typeIdPropName)
{
    // ok to have null/empty; will restore to use defaults
    if (typeIdPropName == null || typeIdPropName.length() == 0) {
        typeIdPropName = _idType.getDefaultPropertyName();
    }
    _typeProperty = typeIdPropName;
    return this;
}

/*
/**********************************************************
/* Accessors
/**********************************************************
 */

public String getTypeProperty() { return _typeProperty; }

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

/**
 * Helper method that will either return configured custom
 * type id resolver, or construct a standard resolver
 * given configuration.
 */
protected TypeIdResolver idResolver(JavaType baseType, Collection<NamedType> subtypes,
        boolean forSer, boolean forDeser)
{
    // Custom id resolver?
    if (_customIdResolver != null) {
        return _customIdResolver;
    }
```

```
        if (_idType == null) {
            throw new IllegalStateException("Can not build, 'init()' not yet called");
        }
        switch (_idType) {
        case CLASS:
            return new ClassNameIdResolver(baseType);
        case MINIMAL_CLASS:
            return new MinimalClassNameIdResolver(baseType);
        case NAME:
            return TypeNameIdResolver.construct(baseType, subtypes, forSer, forDeser);

        case CUSTOM: // need custom resolver...
        case NONE: // hmmh. should never get this far with 'none'
        }
        throw new IllegalStateException("Do not know how to construct standard type id resolver for idType:
"+_idType);
    }
}
package org.codehaus.jackson.map.jsontype.impl;

import org.codehaus.jackson.map.jsontype.TypeIdResolver;
import org.codehaus.jackson.type.JavaType;

public abstract class TypeIdResolverBase
    implements TypeIdResolver
{
    protected final JavaType _baseType;

    protected TypeIdResolverBase(JavaType baseType)
    {
        _baseType = baseType;
    }

    public void init(JavaType bt) {
        /* Standard type id resolvers do not need this;
         * only useful for custom ones.
         */
    }
}
package org.codehaus.jackson.map.jsontype.impl;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.annotate.JsonTypeInfo.As;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;
import org.codehaus.jackson.type.JavaType;
```

```
/**
* Type deserializer used with {@link As#WRAPPER_ARRAY}
* inclusion mechanism. Simple since JSON structure used is always
* the same, regardless of structure used for actual value: wrapping
* is done using a 2-element JSON Array where type id is the first
* element, and actual object data as second element.
*
* @author tatus
*/
public class AsArrayTypeDeserializer extends TypeDeserializerBase
{
  public AsArrayTypeDeserializer(JavaType bt, TypeIdResolver idRes, BeanProperty property)
  {
    super(bt, idRes, property);
  }

  @Override
  public As getTypeInclusion() {
    return As.WRAPPER_ARRAY;
  }

  /**
   * Method called when actual object is serialized as JSON Array.
   */
  @Override
  public Object deserializeTypedFromArray(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
    return _deserialize(jp, ctxt);
  }

  /**
   * Method called when actual object is serialized as JSON Object
   */
  @Override
  public Object deserializeTypedFromObject(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
    return _deserialize(jp, ctxt);
  }

  @Override
  public Object deserializeTypedFromScalar(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
  {
    return _deserialize(jp, ctxt);
  }
```

```
@Override
public Object deserializeTypedFromAny(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   return _deserialize(jp, ctxt);
}

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

/**
 * Method that handles type information wrapper, locates actual
 * subtype deserializer to use, and calls it to do actual
 * deserialization.
 */
private final Object _deserialize(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   JsonDeserializer<Object> deser = _findDeserializer(ctxt, _locateTypeId(jp, ctxt));
   Object value = deser.deserialize(jp, ctxt);
   // And then need the closing END_ARRAY
   if (jp.nextToken() != JsonToken.END_ARRAY) {
      throw ctxt.wrongTokenException(jp, JsonToken.END_ARRAY,
            "expected closing END_ARRAY after type information and deserialized value");
   }
   return value;
}

protected final String _locateTypeId(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   if (!jp.isExpectedStartArrayToken()) {
      throw ctxt.wrongTokenException(jp, JsonToken.START_ARRAY,
            "need JSON Array to contain As.WRAPPER_ARRAY type information for class "+baseTypeName());
   }
   // And then type id as a String
   if (jp.nextToken() != JsonToken.VALUE_STRING) {
      throw ctxt.wrongTokenException(jp, JsonToken.VALUE_STRING,
            "need JSON String that contains type id (for subtype of "+baseTypeName()+")");
   }
   String result = jp.getText();
   jp.nextToken();
   return result;
}
```

```
}
package org.codehaus.jackson.map.jsontype.impl;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.annotate.JsonTypeInfo.As;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;
import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.util.JsonParserSequence;
import org.codehaus.jackson.util.TokenBuffer;

/**
* Type deserializer used with {@link As#PROPERTY}
* inclusion mechanism.
* Uses regular form (additional key/value entry before actual data)
* when typed object is expressed as JSON Object; otherwise behaves similar to how
* {@link As#WRAPPER_ARRAY} works.
* Latter is used if JSON representation is polymorphic
*
* @since 1.5
* @author tatu
*/
public class AsPropertyTypeDeserializer extends AsArrayTypeDeserializer
{
  protected final String _typePropertyName;

  public AsPropertyTypeDeserializer(JavaType bt, TypeIdResolver idRes, BeanProperty property,
        String typePropName)
  {
     super(bt, idRes, property);
     _typePropertyName = typePropName;
  }

  @Override
  public As getTypeInclusion() {
     return As.PROPERTY;
  }

  @Override
  public String getPropertyName() { return _typePropertyName; }

  /**
   * This is the trickiest thing to handle, since property we are looking
   * for may be anywhere...
   */
  @Override
```

```
public Object deserializeTypedFromObject(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
{
   // but first, sanity check to ensure we have START_OBJECT or FIELD_NAME
   JsonToken t = jp.getCurrentToken();
   if (t == JsonToken.START_OBJECT) {
      t = jp.nextToken();
   } else if (t != JsonToken.FIELD_NAME) {
      throw ctxt.wrongTokenException(jp, JsonToken.START_OBJECT,
            "need JSON Object to contain As.PROPERTY type information (for class "+baseTypeName()+")");
   }
   // Ok, let's try to find the property. But first, need token buffer...
   TokenBuffer tb = null;

   for (; t == JsonToken.FIELD_NAME; t = jp.nextToken()) {
      String name = jp.getCurrentName();
      jp.nextToken(); // to point to the value
      if (_typePropertyName.equals(name)) { // gotcha!
         JsonDeserializer<Object> deser = _findDeserializer(ctxt, jp.getText());
         // deserializer should take care of closing END_OBJECT as well
        if (tb != null) {
            jp = JsonParserSequence.createFlattened(tb.asParser(jp), jp);
         }
         /* Must point to the next value; tb had no current, jp
          * pointed to VALUE_STRING:
          */
         jp.nextToken(); // to skip past String value
         // deserializer should take care of closing END_OBJECT as well
         return deser.deserialize(jp, ctxt);
      }
      if (tb == null) {
         tb = new TokenBuffer(null);
      }
      tb.writeFieldName(name);
      tb.copyCurrentStructure(jp);
   }
   // Error if we get here...
   throw ctxt.wrongTokenException(jp, JsonToken.FIELD_NAME,
         "missing property '"+_typePropertyName+"' that is to contain type id  (for class "+baseTypeName()+")");
}

/* As per [JACKSON-352], also need to re-route "unknown" version. Need to think
 * this through bit more in future, but for now this does address issue and has
 * no negative side effects (at least within existing unit test suite).
 */
@Override
public Object deserializeTypedFromAny(JsonParser jp, DeserializationContext ctxt)
   throws IOException, JsonProcessingException
```

```
  {
    /* [JACKSON-387]: Sometimes, however, we get an array wrapper; specifically
     *   when an array or list has been serialized with type information.
     */
    if (jp.getCurrentToken() == JsonToken.START_ARRAY) {
      return super.deserializeTypedFromArray(jp, ctxt);
    }
    return deserializeTypedFromObject(jp, ctxt);
  }

  // These are fine from base class:
  //public Object deserializeTypedArray(JsonParser jp, DeserializationContext ctxt)
  //public Object deserializeTypedScalar(JsonParser jp, DeserializationContext ctxt)
}
package org.codehaus.jackson.map.jsontype.impl;

import java.util.*;

import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.annotate.JsonTypeInfo;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.map.util.ClassUtil;

public class ClassNameIdResolver
  extends TypeIdResolverBase
{
  public ClassNameIdResolver(JavaType baseType) {
    super(baseType);
  }

  public JsonTypeInfo.Id getMechanism() { return JsonTypeInfo.Id.CLASS; }

  public void registerSubtype(Class<?> type, String name) {
    // not used with class name - based resolvers
  }

  public String idFromValue(Object value)
  {
    Class<?> cls = value.getClass();

    // [JACKSON-380] Need to ensure that "enum subtypes" work too
    if (Enum.class.isAssignableFrom(cls)) {
      if (!cls.isEnum()) { // means that it's sub-class of base enum, so:
        cls = cls.getSuperclass();
      }
    }
    String str = cls.getName();
    if (str.startsWith("java.util")) {
```

```
            /* 25-Jan-2009, tatus: There are some internal classes that
             *   we can not access as is. We need better mechanism; for
             *   now this has to do...
             */
            /* Enum sets and maps are problematic since we MUST know
             * type of contained enums, to be able to deserialize.
             * In addition, EnumSet is not a concrete type either
             */
            if (value instanceof EnumSet<?>) { // Regular- and JumboEnumSet...
                Class<?> enumClass = ClassUtil.findEnumType((EnumSet<?>) value);
                str = TypeFactory.collectionType(EnumSet.class, enumClass).toCanonical();
            } else if (value instanceof EnumMap<?,?>) {
                Class<?> enumClass = ClassUtil.findEnumType((EnumMap<?,?>) value);
                Class<?> valueClass = Object.class;
                str = TypeFactory.mapType(EnumMap.class, enumClass, valueClass).toCanonical();
            } else {
                String end = str.substring(9);
                if ((end.startsWith(".Arrays$") || end.startsWith(".Collections$"))
                        && str.indexOf("List") >= 0) {
                    /* 17-Feb-2010, tatus: Another such case: result of
                     *    Arrays.asList() is named like so in Sun JDK...
                     *   Let's just plain old ArrayList in its place
                     * NOTE: chances are there are plenty of similar cases
                     * for other wrappers... (immutable, singleton, synced etc)
                     */
                    str = "java.util.ArrayList";
                }
            }
        }
        return str;
    }

    public JavaType typeFromId(String id)
    {
        /* 30-Jan-2010, tatu: Most ids are basic class names; so let's first
         *    check if any generics info is added; and only then ask factory
         *    to do translation when necessary
         */
        if (id.indexOf('<') > 0) {
            JavaType t = TypeFactory.fromCanonical(id);
            // note: may want to try combining with specialization (esp for EnumMap)
            return t;
        }
        try {
            /* [JACKSON-350]: Default Class.forName() won't work too well; context class loader
             *    seems like slightly better choice
             */
//          Class<?> cls = Class.forName(id);
```

```
            ClassLoader loader = Thread.currentThread().getContextClassLoader();
            Class<?> cls = Class.forName(id, true, loader);
            return TypeFactory.specialize(_baseType, cls);
        } catch (ClassNotFoundException e) {
            throw new IllegalArgumentException("Invalid type id '"+id+"' (for id type 'Id.class'): no such class found");
        } catch (Exception e) {
            throw new IllegalArgumentException("Invalid type id '"+id+"' (for id type 'Id.class'): "+e.getMessage(), e);
        }
    }
}
package org.codehaus.jackson.map.jsontype.impl;

import java.io.IOException;

import org.codehaus.jackson.JsonGenerator;
import org.codehaus.jackson.JsonProcessingException;
import org.codehaus.jackson.annotate.JsonTypeInfo.As;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;

/**
* Type serializer that preferably embeds type information as an additional
* JSON Object property, if possible (when resulting serialization would
* use JSON Object). If this is not possible (for JSON Arrays, scalars),
* uses a JSON Array wrapper (similar to how
* {@link As#WRAPPER_ARRAY} always works) as a fallback.
*
* @since 1.5
* @author tatus
*/
public class AsPropertyTypeSerializer
    extends AsArrayTypeSerializer
{
    protected final String _typePropertyName;

    public AsPropertyTypeSerializer(TypeIdResolver idRes, BeanProperty property,
            String propName)
    {
        super(idRes, property);
        _typePropertyName = propName;
    }

    @Override
    public String getPropertyName() { return _typePropertyName; }

    @Override
    public As getTypeInclusion() { return As.PROPERTY; }
```

```
  @Override
  public void writeTypePrefixForObject(Object value, JsonGenerator jgen)
     throws IOException, JsonProcessingException
  {
     jgen.writeStartObject();
     jgen.writeStringField(_typePropertyName, _idResolver.idFromValue(value));
  }

  //public void writeTypePrefixForArray(Object value, JsonGenerator jgen)
  //public void writeTypePrefixForScalar(Object value, JsonGenerator jgen)

  @Override
  public void writeTypeSuffixForObject(Object value, JsonGenerator jgen)
     throws IOException, JsonProcessingException
  {
     jgen.writeEndObject();
  }

  //public void writeTypeSuffixForArray(Object value, JsonGenerator jgen)
  //public void writeTypeSuffixForScalar(Object value, JsonGenerator jgen)
}
package org.codehaus.jackson.map.jsontype.impl;

import org.codehaus.jackson.type.JavaType;
import org.codehaus.jackson.annotate.JsonTypeInfo;

public class MinimalClassNameIdResolver
  extends ClassNameIdResolver
{
  /**
   * Package name of the base class, to be used for determining common
   * prefix that can be omitted from included type id.
   * Does not include the trailing dot.
   */
  protected final String _basePackageName;

  /**
   * Same as {@link #_basePackageName}, but includes trailing dot.
   */
  protected final String _basePackagePrefix;

  protected MinimalClassNameIdResolver(JavaType baseType)
  {
     super(baseType);
     String base = baseType.getRawClass().getName();
     int ix = base.lastIndexOf('.');
     if (ix < 0) { // can this ever occur?
        _basePackageName = "";
```

```
            _basePackagePrefix = ".";
        } else {
            _basePackagePrefix = base.substring(0, ix+1);
            _basePackageName = base.substring(0, ix);
        }
    }

    @Override
    public JsonTypeInfo.Id getMechanism() { return JsonTypeInfo.Id.MINIMAL_CLASS; }

    @Override
    public String idFromValue(Object value)
    {
        String n = value.getClass().getName();
        if (n.startsWith(_basePackagePrefix)) {
            // note: we will leave the leading dot in there
            return n.substring(_basePackagePrefix.length()-1);
        }
        return n;
    }

    @Override
    public JavaType typeFromId(String id)
    {
        if (id.startsWith(".")) {
            StringBuilder sb = new StringBuilder(id.length() + _basePackageName.length());
            if  (_basePackageName.length() == 0) {
                // no package; must remove leading '.' from id
                sb.append(id.substring(1));
            } else {
                // otherwise just concatenate package, with leading-dot-partial name
                sb.append(_basePackageName).append(id);
            }
            id = sb.toString();
        }
        return super.typeFromId(id);
    }
}
package org.codehaus.jackson.map.jsontype.impl;

import java.util.*;

import org.codehaus.jackson.annotate.JsonTypeInfo;
import org.codehaus.jackson.map.jsontype.NamedType;
import org.codehaus.jackson.map.type.TypeFactory;
import org.codehaus.jackson.type.JavaType;

public class TypeNameIdResolver
```

```
  extends TypeIdResolverBase
{
  /**
   * Mappings from class name to type id, used for serialization
   */
  protected final HashMap<String, String> _typeToId;

  /**
   * Mappings from type id to JavaType, used for deserialization
   */
  protected final HashMap<String, JavaType> _idToType;

  protected TypeNameIdResolver(JavaType baseType,
         HashMap<String, String> typeToId,
         HashMap<String, JavaType> idToType)
  {
     super(baseType);
     _typeToId = typeToId;
     _idToType = idToType;
  }

  public static TypeNameIdResolver construct(JavaType baseType,
         Collection<NamedType> subtypes, boolean forSer, boolean forDeser)
  {
     // sanity check
     if (forSer == forDeser) throw new IllegalArgumentException();
     HashMap<String, String> typeToId = null;
     HashMap<String, JavaType> idToType = null;

     if (forSer) {
        typeToId = new HashMap<String, String>();
     }
     if (forDeser) {
        idToType = new HashMap<String, JavaType>();
     }
     if (subtypes != null) {
        for (NamedType t : subtypes) {
           /* no name? Need to figure out default; for now, let's just
            * use non-qualified class name
            */
           Class<?> cls = t.getType();
           String id = t.hasName() ? t.getName() : _defaultTypeId(cls);
           if (forSer) {
              typeToId.put(cls.getName(), id);
           }
           if (forDeser) {
              /* 24-Feb-2011, tatu: [JACKSON-498] One more problem; sometimes
               *   we have same name for multiple types; if so, use most specific
```

```
             *   one.
             */
            JavaType prev = idToType.get(id);
            if (prev != null) { // Can only override if more specific
               if (cls.isAssignableFrom(prev.getRawClass())) { // nope, more generic (or same)
                  continue;
               }
            }
            idToType.put(id, TypeFactory.type(cls));
         }
      }
   }
   return new TypeNameIdResolver(baseType, typeToId, idToType);
}

public JsonTypeInfo.Id getMechanism() { return JsonTypeInfo.Id.NAME; }

public String idFromValue(Object value)
{
   Class<?> cls = value.getClass();
   String name = _typeToId.get(cls.getName());
   // can either throw an exception, or use default name...
   if (name == null) {
      // let's choose default?
      name = _defaultTypeId(cls);
   }
   return name;
}

public JavaType typeFromId(String id)
   throws IllegalArgumentException
{
   JavaType t = _idToType.get(id);
   /* Now: if no type is found, should we try to locate it by
    * some other means? (specifically, if in same package as base type,
    * could just try Class.forName)
    * For now let's not add any such workarounds; can add if need be
    */
   return t;
}

@Override
public String toString()
{
 StringBuilder sb = new StringBuilder();
 sb.append('[').append(getClass().getName());
 sb.append("; id-to-type=").append(_idToType);
 sb.append(']');
```

```
    return sb.toString();
  }

  /*
  /**********************************************************
  /* Helper methods
  /**********************************************************
   */

  /**
   * If no name was explicitly given for a class, we will just
   * use non-qualified class name
   */
  protected static String _defaultTypeId(Class<?> cls)
  {
      String n = cls.getName();
      int ix = n.lastIndexOf('.');
      return (ix < 0) ? n : n.substring(ix+1);
  }
}
package org.codehaus.jackson.map.jsontype.impl;

import java.io.IOException;

import org.codehaus.jackson.*;
import org.codehaus.jackson.annotate.JsonTypeInfo.As;
import org.codehaus.jackson.map.*;
import org.codehaus.jackson.map.jsontype.TypeIdResolver;
import org.codehaus.jackson.type.JavaType;

/**
* Type deserializer used with {@link As#WRAPPER_OBJECT}
* inclusion mechanism. Simple since JSON structure used is always
* the same, regardless of structure used for actual value: wrapping
* is done using a single-element JSON Object where type id is the key,
* and actual object data as the value.
*
* @author tatus
*/
public class AsWrapperTypeDeserializer extends TypeDeserializerBase
{
  public AsWrapperTypeDeserializer(JavaType bt, TypeIdResolver idRes, BeanProperty property)
  {
      super(bt, idRes, property);
  }

  @Override
  public As getTypeInclusion() {
```

```
    return As.WRAPPER_OBJECT;
}

/**
 * Deserializing type id enclosed using WRAPPER_OBJECT style is straightforward
 */
@Override
public Object deserializeTypedFromObject(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    return _deserialize(jp, ctxt);
}

@Override
public Object deserializeTypedFromArray(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    return _deserialize(jp, ctxt);
}

@Override
public Object deserializeTypedFromScalar(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    return _deserialize(jp, ctxt);
}

@Override
public Object deserializeTypedFromAny(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
    return _deserialize(jp, ctxt);
}

/*
/**********************************************************
/* Internal methods
/**********************************************************
 */

/**
 * Method that handles type information wrapper, locates actual
 * subtype deserializer to use, and calls it to do actual
 * deserialization.
 */
private final Object _deserialize(JsonParser jp, DeserializationContext ctxt)
    throws IOException, JsonProcessingException
{
```

```
        // first, sanity checks
        if (jp.getCurrentToken() != JsonToken.START_OBJECT) {
            throw ctxt.wrongTokenException(jp, JsonToken.START_OBJECT,
                    "need JSON Object to contain As.WRAPPER_OBJECT type information for class "+baseTypeName());
        }
        // should always get field name, but just in case...
        if (jp.nextToken() != JsonToken.FIELD_NAME) {
            throw ctxt.wrongTokenException(jp, JsonToken.FIELD_NAME,
                    "need JSON String that contains type id (for subtype of "+baseTypeName()+")");
        }
        JsonDeserializer<Object> deser = _findDeserializer(ctxt, jp.getText());
        jp.nextToken();
        Object value = deser.deserialize(jp, ctxt);
        // And then need the closing END_OBJECT
        if (jp.nextToken() != JsonToken.END_OBJECT) {
            throw ctxt.wrongTokenException(jp, JsonToken.END_OBJECT,
                    "expected closing END_OBJECT after type information and deserialized value");
        }
        return value;
    }
}
package org.codehaus.jackson.map.jsontype;

import java.util.Collection;

import org.codehaus.jackson.map.AnnotationIntrospector;
import org.codehaus.jackson.map.MapperConfig;
import org.codehaus.jackson.map.introspect.AnnotatedClass;
import org.codehaus.jackson.map.introspect.AnnotatedMember;

/**
* Helper object used for handling registration on resolving of supertypes
* to subtypes.
*/
public abstract class SubtypeResolver
{
    /**
     * Method for registering specified subtypes (possibly including type
     * names); for type entries without name, non-qualified class name
     * as used as name (unless overridden by annotation).
     */
    public abstract void registerSubtypes(NamedType... types);

    public abstract void registerSubtypes(Class<?>... classes);

    /**
     * Method for finding out all reachable subtypes for a property specified
     * by given element (method or field)
```

```
     */
    public abstract Collection<NamedType> collectAndResolveSubtypes(AnnotatedMember property,
            MapperConfig<?> config, AnnotationIntrospector ai);

    /**
     * Method for finding out all reachable subtypes for given type.
     */
    public abstract Collection<NamedType> collectAndResolveSubtypes(AnnotatedClass basetype,
            MapperConfig<?> config, AnnotationIntrospector ai);
}
package org.codehaus.jackson.map.jsontype;

import org.codehaus.jackson.annotate.JsonTypeInfo;
import org.codehaus.jackson.type.JavaType;

/**
* Interface that defines standard API for converting types
* to type identifiers and vice versa. Used by type resolvers
* ({@link org.codehaus.jackson.map.TypeSerializer},
* {@link org.codehaus.jackson.map.TypeDeserializer}) for converting
* between type and matching id; id is stored in JSON and needed for
* creating instances of proper subtypes when deserializing values.
*
* @author tatu
* @since 1.5
*/
public interface TypeIdResolver
{
    /*
    /**********************************************************
    /* Initialization/configuration methods
    /**********************************************************
     */

    /**
     * Method that will be called once before any type resolution calls;
     * used to initialize instance with configuration. This is necessary
     * since instances may be created via reflection, without ability to
     * call specific constructor to pass in configuration settings.
     *
     * @param baseType Base type for which this id resolver instance is
     *   used
     */
    public void init(JavaType baseType);

    /*
    /**********************************************************
    /* Conversions between types and type ids
```

```
  /**********************************************************
   */

  /**
   * Method called to serialize type of the type of given value
   * as a String to include in serialized JSON content.
   */
  public String idFromValue(Object value);

  /**
   * Method called to resolve type from given type identifier.
   */
  public JavaType typeFromId(String id);

  /*
  /**********************************************************
  /* Accessors for metadata
  /**********************************************************
   */

   /**
    * Accessor for mechanism that this resolver uses for determining
    * type id from type. Mostly informational; not required to be called
    * or used.
    */
   public JsonTypeInfo.Id getMechanism();
}
package org.codehaus.jackson.map.jsontype;

import java.util.Collection;

import org.codehaus.jackson.annotate.JsonTypeInfo;
import org.codehaus.jackson.annotate.JsonTypeInfo.As;
import org.codehaus.jackson.map.BeanProperty;
import org.codehaus.jackson.map.TypeDeserializer;
import org.codehaus.jackson.map.TypeSerializer;
import org.codehaus.jackson.type.JavaType;

/**
* Interface that defines builders that are configured based on
* annotations (like {@link JsonTypeInfo} or JAXB annotations),
* and produce type serializers and deserializers used for
* handling type information embedded in JSON to allow for safe
* polymorphic type handling.
*<p>
* Builder is first initialized by calling {@link #init} method, and then
* configured using <code>setXxx</code> (and <code>registerXxx</code>)
* methods. Finally, after calling all configuration methods,
```

```
* {@link #buildTypeSerializer} or {@link #buildTypeDeserializer}
* will be called to get actual type resolver constructed
* and used for resolving types for configured base type and its
* subtypes.
*
* @since 1.5
* @author tatu
*/
public interface TypeResolverBuilder<T extends TypeResolverBuilder<T>>
{
  /*
  /**********************************************************
  /* Actual builder methods
  /**********************************************************
   */

  /**
   * Method for building type serializer based on current configuration
   * of this builder.
   *
   * @param baseType Base type that constructed resolver will
   *    handle; super type of all types it will be used for.
   */
  public TypeSerializer buildTypeSerializer(JavaType baseType, Collection<NamedType> subtypes,
        BeanProperty property);

  /**
   * Method for building type deserializer based on current configuration
   * of this builder.
   *
   * @param baseType Base type that constructed resolver will
   *    handle; super type of all types it will be used for.
   * @param subtypes Known subtypes of the base type.
   */
  public TypeDeserializer buildTypeDeserializer(JavaType baseType,
        Collection<NamedType> subtypes, BeanProperty property);

  /*
  /**********************************************************
  /* Initialization method(s) that must be called before other
  /* configuration
  /**********************************************************
   */

  /**
   * Initialization method that is called right after constructing
   * the builder instance.
   *
```

```
     * @param idType Which type metadata is used
     * @param res (optional) Custom type id resolver used, if any
     *
     * @return Resulting builder instance (usually this builder,
     *   but not necessarily)
     */
    public T init(JsonTypeInfo.Id idType, TypeIdResolver res);

    /*
    /**********************************************************
    /* Methods for configuring resolver to build
    /**********************************************************
     */

    /**
     * Method for specifying mechanism to use for including type metadata
     * in JSON.
     * If not explicitly called, setting defaults to
     * {@link As#PROPERTY}.
     *
     * @param includeAs Mechanism used for including type metadata in JSON
     *
     * @return Resulting builder instance (usually this builder,
     *   but not necessarily)
     */
    public T inclusion(As includeAs);

    /**
     * Method for specifying name of property used for including type
     * information. Not used for all inclusiong mechanisms;
     * usually only used with {@link As#PROPERTY}.
     *<p>
     * If not explicitly called, name of property to use is based on
     * defaults for {@link org.codehaus.jackson.annotate.JsonTypeInfo.Id} configured.
     *
     * @param propName Name of JSON property to use for including
     *    type information
     *
     * @return Resulting builder instance (usually this builder,
     *   but not necessarily)
     */
    public T typeProperty(String propName);
}
package org.codehaus.jackson.map.jsontype;

/**
* Simple container class for types with optional logical name, used
* as external identifier
```

```
*
* @author tatu
* @since 1.5
*/
public final class NamedType
{
  protected final Class<?> _class;
  protected final int _hashCode;

  protected String _name;

  public NamedType(Class<?> c) { this(c, null); }

  public NamedType(Class<?> c, String name)
  {
     _class = c;
     _hashCode = c.getName().hashCode();
     setName(name);
  }

  public Class<?> getType() { return _class; }
  public String getName() { return _name; }
  public void setName(String name) {
     _name = (name == null || name.length() == 0) ? null : name;
  }

  public boolean hasName() { return _name != null; }

  /**
   * Equality is defined based on class only, not on name
   */
  @Override
  public boolean equals(Object o)
  {
     if (o == this) return true;
     if (o == null) return false;
     if (o.getClass() != getClass()) return false;
     return _class == ((NamedType) o)._class;
  }

  @Override
  public int hashCode() { return _hashCode; }

  @Override
  public String toString() {
   return "[NamedType, class "+_class.getName()+", name: "+(_name == null ? "null" :(""+_name+""))+"]";
  }
}
```

```
/* Jackson JSON-processor.
*
* Copyright (c) 2007- Tatu Saloranta, tatu.saloranta@iki.fi
*
* Licensed under the License specified in file LICENSE, included with
* the source code and binary code bundles.
* You may not use this file except in compliance with the License.
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package org.codehaus.jackson.map;

import org.codehaus.jackson.*;

/**
* Sub-class of {@link JsonFactory} that will create a proper
* {@link ObjectCodec} to allow seamless conversions between
* Json content and Java objects (POJOs).
* The only addition to regular {@link JsonFactory} currently
* is that {@link ObjectMapper} is constructed and passed as
* the codec to use.
*/
public class MappingJsonFactory
  extends JsonFactory
{
  public MappingJsonFactory()
  {
     this(null);
  }

  public MappingJsonFactory(ObjectMapper mapper)
  {
     super(mapper);
     if (mapper == null) {
        setCodec(new ObjectMapper(this));
     }
  }

  /**
   * We'll override the method to return more specific type; co-variance
   * helps here
   */
  @Override
  public final ObjectMapper getCodec() { return (ObjectMapper) _objectCodec; }
```

```
}
package org.codehaus.jackson.map;

/**
* Add-on interface that {@link JsonDeserializer}s can implement to get a callback
* that can be used to create contextual instances of deserializer to use for
* handling properties of supported type. This can be useful
* for deserializers that can be configured by annotations, or should otherwise
* have differing behavior depending on what kind of property is being deserialized.
*
* @param <T> Type of deserializer to contextualize
*
* @since 1.7
*/
public interface ContextualDeserializer<T>
{
  /**
   * Method called to see if a different (or differently configured) deserializer
   * is needed to deserialize values of specified property.
   * Note that instance that this method is called on is typically shared one and
   * as a result method should <b>NOT</b> modify this instance but rather construct
   * and return a new instance. This instance should only be returned as-is, in case
   * it is already suitable for use.
   *
   * @param config Current deserialization configuration
   * @param property Method, field or constructor parameter that represents the property
   *   (and is used to assign deserialized value).
   *   Should be available; but there may be cases where caller can not provide it and
   *   null is passed instead (in which case impls usually pass 'this' deserializer as is)
   *
   * @return Deserializer to use for deserializing values of specified property;
   *   may be this instance or a new instance.
   *
   * @throws JsonMappingException
   */
  public JsonDeserializer<T> createContextual(DeserializationConfig config,
        BeanProperty property)
    throws JsonMappingException;
}
/**
* Classes needed for JSON schema support (currently just ability
* to generate schemas using serialization part of data mapping)
*/
package org.codehaus.jackson.schema;
package org.codehaus.jackson.schema;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.annotate.JsonCreator;
```

```
import org.codehaus.jackson.annotate.JsonValue;
import org.codehaus.jackson.node.ObjectNode;
import org.codehaus.jackson.node.JsonNodeFactory;

/**
* A {@link org.codehaus.jackson.JsonNode} that represents a JSON-Schema instance.
*
* @author Ryan Heaton
* @see <a href="http://json-schema.org/">JSON Schema</a>
*/
public class JsonSchema
{
  private final ObjectNode schema;

  /**
   * Main constructor for schema instances.
   *<p>
   * This is the creator constructor used by Jackson itself when
   * deserializing instances. It is so-called delegating creator,
   * meaning that its argument will be bound by Jackson before
   * constructor gets called.
   */
  @JsonCreator
  public JsonSchema(ObjectNode schema)
  {
     this.schema = schema;
  }

  /**
   * Method for accessing root JSON object of the contained schema.
   *<p>
   * Note: this method is specified with {@link JsonValue} annotation
   * to represent serialization to use; same as if explicitly
   * serializing returned object.
   *
   * @return Root node of the schema tree
   */
  @JsonValue
  public ObjectNode getSchemaNode()
  {
     return schema;
  }

  @Override
  public String toString()
  {
     return this.schema.toString();
  }
```

```
  @Override
  public boolean equals(Object o)
  {
     if (o == this) return true;
     if (o == null) return false;
     if (!(o instanceof JsonSchema)) return false;

     JsonSchema other = (JsonSchema) o;
     if (schema == null) {
        return other.schema == null;
     }
     return schema.equals(other.schema);
  }

  /**
   * Get the default schema node.
   *
   * @return The default schema node.
   */
  public static JsonNode getDefaultSchemaNode()
  {
     ObjectNode objectNode = JsonNodeFactory.instance.objectNode();
     objectNode.put("type", "any");
     objectNode.put("optional", true);
     return objectNode;
  }

}
package org.codehaus.jackson.schema;

import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Retention;
import java.lang.annotation.ElementType;
import java.lang.annotation.Target;

import org.codehaus.jackson.annotate.JacksonAnnotation;

/**
* Annotation that can be used to define JSON Schema definition for
* the annotated class.
*<p>
* Note that annotation is often not needed: for example, regular
* Jackson beans that Jackson can introspect can be used without
* annotations, to produce JSON schema definition.
*
* @author Ryan Heaton
*/
```

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonSerializableSchema
{
  /**
   * The schema type for this JsonSerializable instance.
   * Possible values: "string", "number", "boolean", "object", "array", "null", "any"
   *
   * @return The schema type for this JsonSerializable instance.
   */
  String schemaType() default "any";

  /**
   * If the schema type is "object", the node that defines the properties of the object.
   *
   * @return The node representing the schema properties, or "##irrelevant" if irrelevant.
   */
  String schemaObjectPropertiesDefinition() default "##irrelevant";

  /**
   * If the schema type if "array", the node that defines the schema for the items in the array.
   *
   * @return The schema for the items in the array, or "##irrelevant" if irrelevant.
   */
  String schemaItemDefinition() default "##irrelevant";
}
package org.codehaus.jackson.schema;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.SerializerProvider;
import org.codehaus.jackson.map.JsonMappingException;

import java.lang.reflect.Type;

/**
* Marker interface for schema-aware serializers.
*
* @author Ryan Heaton
*/
public interface SchemaAware
{
  /**
   * Get the representation of the schema to which this serializer will conform.
   *
   * @param provider The serializer provider.
   * @param typeHint A hint about the type.
   * @return <a href="http://json-schema.org/">Json-schema</a> for this serializer.
```

```
     */
    JsonNode getSchema(SerializerProvider provider, Type typeHint)
            throws JsonMappingException;
}
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.8.2
Created-By: 1.6.0_22-b04-307-10M3261 (Apple Inc.)
```

1.81 jarjar_within-cglib 1.0rc8

1.81.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a

copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity
on behalf of whom a Contribution has been received by Licensor and
subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
copyright license to reproduce, prepare Derivative Works of,
publicly display, publicly perform, sublicense, and distribute the
Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
(except as stated in this section) patent license to make, have made,
use, offer to sell, sell, import, and otherwise transfer the Work,
where such license applies only to those patent claims licensable
by such Contributor that are necessarily infringed by their
Contribution(s) alone or by combination of their Contribution(s)
with the Work to which such Contribution(s) was submitted. If You
institute patent litigation against any entity (including a
cross-claim or counterclaim in a lawsuit) alleging that the Work
or a Contribution incorporated within the Work constitutes direct

or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of

this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following

boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

```
/**
* Copyright 2007 Google Inc.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
```

1.82 javacsv 2.0

1.82.1 Available under license :

* Java CSV is a stream based library for reading and writing

* CSV and other delimited data.
*
* Copyright (C) Bruce Dunwiddie bruce@csvreader.com

*

* This library is free software; you can redistribute it and/or
* modify it under the terms of the GNU Lesser General Public

* License as published by the Free Software Foundation; either

* version 2.1 of the License, or (at your option) any later version.

*
* This library is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of

* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

* Lesser General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public

* License along with this library; if not, write to the Free Software

* Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA
GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]

Preamble

The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some
specially designated software packages--typically libraries--of the
Free Software Foundation and other authors who decide to use it. You
can use it too, but we suggest you first think carefully about whether
this license or the ordinary General Public License is the better
strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use,
not price. Our General Public Licenses are designed to make sure that

you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the

entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data

prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no

charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for

that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6,

whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library `Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice

That's all there is to it!

1.83 javassist 3.16.1

1.83.1 Available under license :

```
/*
* Javassist, a Java-bytecode translator toolkit.
* Copyright (C) 1999- Shigeru Chiba. All Rights Reserved.
*
* The contents of this file are subject to the Mozilla Public License Version
* 1.1 (the "License"); you may not use this file except in compliance with
* the License.  Alternatively, the contents of this file may be used under
* the terms of the GNU Lesser General Public License Version 2.1 or later,
* or the Apache License Version 2.0.
*
* Software distributed under the License is distributed on an "AS IS" basis,
* WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License
* for the specific language governing rights and limitations under the
* License.
*/
```

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical

transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable

by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use,

reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.84 javax.jms-api 2.0

1.84.1 Available under license :

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. Contributor. means each individual or entity that creates or contributes to the creation of Modifications.

1.2. Contributor Version. means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. Covered Software. means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. Executable. means the Covered Software in any form other than Source Code.

1.5. Initial Developer. means the individual or entity that first makes Original Software available under this License.

1.6. Larger Work. means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. License. means this document.

1.8. Licensable. means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications. means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. Original Software. means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. Patent Claims. means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. Source Code. means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. You. (or .Your.) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, .You. includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, .control. means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or

otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the

Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients. rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipient.s rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it

clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN .AS IS. BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as .Participant.) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY.S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT

APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a .commercial item,. as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of .commercial computer software. (as that term is defined at 48 C.F.R. ? 252.227-7014(a)(1)) and .commercial computer software documentation. as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdiction.s conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys. fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)

The code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY

TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

One line to give the program's name and a brief idea of what it does.

Copyright (C)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at

compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

"CLASSPATH" EXCEPTION TO THE GPL VERSION 2

Certain source files distributed by Sun Microsystems, Inc. are subject to the following clarification and special exception to the GPL Version 2, but only where Sun has expressly included in the particular source file's header the words

"Sun designates this particular file as subject to the "Classpath" exception as provided by Sun in the License file that accompanied this code."

Linking this library statically or dynamically with other modules is making a combined work based on this library. Thus, the terms and conditions of the GNU General Public License Version 2 cover the whole combination.

As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module.? An independent module is a module which is not derived from or based on this library.? If you modify this library, you may extend this exception to your version of the library, but you are not obligated to do so.? If you do not wish to do so, delete this exception statement from your version.

1.85 Javax.Servlet-api 3.0.1

1.85.1 Available under license :

/*
* DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.
*
* Copyright 1997-2008 Sun Microsystems, Inc. All rights reserved.
*
* The contents of this file are subject to the terms of either the GNU
* General Public License Version 2 only ("GPL") or the Common Development
* and Distribution License("CDDL") (collectively, the "License"). You
* may not use this file except in compliance with the License. You can obtain
* a copy of the License at https://glassfish.dev.java.net/public/CDDL+GPL.html
* or glassfish/bootstrap/legal/LICENSE.txt. See the License for the specific
* language governing permissions and limitations under the License.
*
* When distributing the software, include this License Header Notice in each

* file and include the License file at glassfish/bootstrap/legal/LICENSE.txt.
* Sun designates this particular file as subject to the "Classpath" exception
* as provided by Sun in the GPL Version 2 section of the License file that
* accompanied this code. If applicable, add the following below the License
* Header, with the fields enclosed by brackets [] replaced by your own
* identifying information: "Portions Copyrighted [year]
* [name of copyright owner]"
*
* Contributor(s):
*
* If you wish your version of this file to be governed by only the CDDL or
* only the GPL Version 2, indicate your decision by adding "[Contributor]
* elects to include this software in this distribution under the [CDDL or GPL
* Version 2] license." If you don't indicate a single choice of license, a
* recipient has the option to distribute your version of this file under
* either the CDDL, the GPL Version 2 or to extend the choice of license to
* its licensees as provided above. However, if you add GPL Version 2 code
* and therefore, elected the GPL Version 2 license, then the option applies
* only if the new code is made subject to such option by the copyright
* holder.
*
*
* This file incorporates work covered by the following copyright and
* permission notice:
*
* Copyright 2004 The Apache Software Foundation
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. Contributor. means each individual or entity that creates or contributes to the creation of Modifications.

1.2. Contributor Version. means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. Covered Software. means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. Executable. means the Covered Software in any form other than Source Code.

1.5. Initial Developer. means the individual or entity that first makes Original Software available under this License.

1.6. Larger Work. means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. License. means this document.

1.8. Licensable. means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications. means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. Original Software. means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. Patent Claims. means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. Source Code. means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. You. (or .Your.) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, .You. includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, .control. means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use,

reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients. rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipient.s rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may

also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN .AS IS. BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as .Participant.) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES

FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY.S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a .commercial item,. as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of .commercial computer software. (as that term is defined at 48 C.F.R. ? 252.227-7014(a)(1)) and .commercial computer software documentation. as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdiction.s conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys. fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)

The code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.

The GNU General Public License (GPL) Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited

to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE

PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

One line to give the program's name and a brief idea of what it does.

Copyright (C)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License.

Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

"CLASSPATH" EXCEPTION TO THE GPL VERSION 2

Certain source files distributed by Sun Microsystems, Inc. are subject to the following clarification and special exception to the GPL Version 2, but only where Sun has expressly included in the particular source file's header the words

"Sun designates this particular file as subject to the "Classpath" exception as provided by Sun in the License file that accompanied this code."

Linking this library statically or dynamically with other modules is making a combined work based on this library. Thus, the terms and conditions of the GNU General Public License Version 2 cover the whole combination.

As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module.? An independent module is a module which is not derived from or based on this library.? If you modify this library, you may extend this exception to your version of the library, but you are not obligated to do so.? If you do not wish to do so, delete this exception statement from your version.

1.86 jaxb-impl 2.2.2

1.86.1 Available under license :

/*
* DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.
*
* Copyright 1997-2007 Sun Microsystems, Inc. All rights reserved.
*
* The contents of this file are subject to the terms of either the GNU
* General Public License Version 2 only ("GPL") or the Common Development

* and Distribution License("CDDL") (collectively, the "License"). You
* may not use this file except in compliance with the License. You can obtain
* a copy of the License at https://glassfish.dev.java.net/public/CDDL+GPL.html
* or glassfish/bootstrap/legal/LICENSE.txt. See the License for the specific
* language governing permissions and limitations under the License.
*
* When distributing the software, include this License Header Notice in each
* file and include the License file at glassfish/bootstrap/legal/LICENSE.txt.
* Sun designates this particular file as subject to the "Classpath" exception
* as provided by Sun in the GPL Version 2 section of the License file that
* accompanied this code. If applicable, add the following below the License
* Header, with the fields enclosed by brackets [] replaced by your own
* identifying information: "Portions Copyrighted [year]
* [name of copyright owner]"
*
* Contributor(s):
*
* If you wish your version of this file to be governed by only the CDDL or
* only the GPL Version 2, indicate your decision by adding "[Contributor]
* elects to include this software in this distribution under the [CDDL or GPL
* Version 2] license." If you don't indicate a single choice of license, a
* recipient has the option to distribute your version of this file under
* either the CDDL, the GPL Version 2 or to extend the choice of license to
* its licensees as provided above. However, if you add GPL Version 2 code
* and therefore, elected the GPL Version 2 license, then the option applies
* only if the new code is made subject to such option by the copyright
* holder.
*/
COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. Contributor means each individual or entity that creates or contributes to the creation of Modifications.

1.2. Contributor Version means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. Covered Software means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. Executable means the Covered Software in any form other than Source Code.

1.5. Initial Developer means the individual or entity that first makes Original Software available under this License.

1.6. Larger Work means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. License means this document.

1.8. Licensable means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. Original Software means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. Patent Claims means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. Source Code means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. You (or Your) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, You includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, control means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipients rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN AS IS BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as Participant) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING

FROM SUCH PARTYS NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a commercial item, as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of commercial computer software (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and commercial computer software documentation as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdictions conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)

The code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307

USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY

TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

One line to give the program's name and a brief idea of what it does.

Copyright (C)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

"CLASSPATH" EXCEPTION TO THE GPL VERSION 2

Certain source files distributed by Sun Microsystems, Inc. are subject to the following clarification and special exception to the GPL Version 2, but only where Sun has expressly included in the particular source file's header the words
"Sun designates this particular file as subject to the "Classpath" exception as provided by Sun in the License file that accompanied this code."

Linking this library statically or dynamically with other modules is making a combined work based on this library. Thus, the terms and conditions of the GNU General Public License Version 2 cover the whole combination.

As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module.? An independent module is a module which is not derived from or based on this library.? If you modify this library, you may extend this exception to your version of the library, but you are not obligated to do so.? If you do not wish to do so, delete this exception statement from your version.

1.87 jaxb-impl-2.2.1.1 2.2.1.1

1.87.1 Available under license :

```
/*
* DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.
*
* Copyright 1997-2007 Sun Microsystems, Inc. All rights reserved.
*
* The contents of this file are subject to the terms of either the GNU
* General Public License Version 2 only ("GPL") or the Common Development
* and Distribution License("CDDL") (collectively, the "License").  You
* may not use this file except in compliance with the License. You can obtain
* a copy of the License at https://glassfish.dev.java.net/public/CDDL+GPL.html
* or glassfish/bootstrap/legal/LICENSE.txt.  See the License for the specific
* language governing permissions and limitations under the License.
*
* When distributing the software, include this License Header Notice in each
* file and include the License file at glassfish/bootstrap/legal/LICENSE.txt.
* Sun designates this particular file as subject to the "Classpath" exception
* as provided by Sun in the GPL Version 2 section of the License file that
```

```
* accompanied this code.  If applicable, add the following below the License
* Header, with the fields enclosed by brackets [] replaced by your own
* identifying information: "Portions Copyrighted [year]
* [name of copyright owner]"
*
* Contributor(s):
*
* If you wish your version of this file to be governed by only the CDDL or
* only the GPL Version 2, indicate your decision by adding "[Contributor]
* elects to include this software in this distribution under the [CDDL or GPL
* Version 2] license."  If you don't indicate a single choice of license, a
* recipient has the option to distribute your version of this file under
* either the CDDL, the GPL Version 2 or to extend the choice of license to
* its licensees as provided above.  However, if you add GPL Version 2 code
* and therefore, elected the GPL Version 2 license, then the option applies
* only if the new code is made subject to such option by the copyright
* holder.
*/

package com.sun.xml.bind;

import java.lang.reflect.Field;
import java.lang.reflect.Method;
import com.sun.xml.bind.v2.runtime.reflect.Accessor;
import javax.xml.bind.JAXBException;

/**
* A means to allow the user to provide customized Accessor
* to be used by JAXB.
*/
public interface AccessorFactory {
  /**
   * Access a field of the class.
   *
   * @param bean the class to be processed.
   * @param f the field within the class to be accessed.
   * @param readOnly  the isStatic value of the field's modifier.
   * @return Accessor the accessor for this field
   *
   * @throws JAXBException reports failures of the method.
   */
  Accessor createFieldAccessor(Class bean, Field f, boolean readOnly) throws JAXBException;

  /**
   * Access a property of the class.
   *
   * @param bean the class to be processed
   * @param getter the getter method to be accessed. The value can be null.
```

```
   * @param setter the setter method to be accessed. The value can be null.
   * @return Accessor the accessor for these methods
   *
   * @throws JAXBException reports failures of the method.
   */
  Accessor createPropertyAccessor(Class bean, Method getter, Method setter) throws JAXBException;
}
```

1.88 jettison 1.3

1.88.1 Available under license :

Copyright 2006 Envoi Solutions LLC

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity

exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided

that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,

or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Copyright 2006 Envoi Solutions LLC

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.89 json-simple 1.1.1

1.89.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or

otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual,

worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents

of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!) The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.90 jsr311-api 1.1.1

1.90.1 Available under license :

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0
[OSI Approved License]

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. "Contributor" means each individual or entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. "Covered Software" means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. "Executable" means the Covered Software in any form other than Source Code.

1.5. "Initial Developer" means the individual or entity that first makes Original Software available under this License.

1.6. "Larger Work" means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. "License" means this document.

1.8. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. "Original Software" means the Source Code and

Executable form of computer software code that is originally released under this License.

1.11. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. "Source Code" means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software

available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the

Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients' rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty,

support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipient's rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the

version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or

a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as "Participant") alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" (as that term is defined at 48 C.F.R.

252.227-7014(a)(1)) and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdiction's conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

1.91 jstl 1.2.0

1.91.1 Available under license :

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. Contributor means each individual or entity that creates or contributes to the creation of Modifications.

1.2. Contributor Version means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. Covered Software means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. Executable means the Covered Software in any form other than Source Code.

1.5. Initial Developer means the individual or entity that first makes Original Software available under this License.

1.6. Larger Work means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. License means this document.

1.8. Licensable means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. Original Software means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. Patent Claims means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. Source Code means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. You (or Your) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, You includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, control means (a)áthe power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b)áownership of more than fifty

percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.
Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:
(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and
(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).
(c) The licenses granted in Sectionsá2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.
(d) Notwithstanding Sectioná2.1(b) above, no patent license is granted: (1)áfor code that You delete from the Original Software, or (2)áfor infringements caused by: (i)áthe modification of the Original Software, or (ii)áthe combination of the Original Software with other software or devices.

2.2. Contributor Grant.
Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:
(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and
(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1)áModifications made by that Contributor (or portions thereof); and (2)áthe combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).
(c) The licenses granted in Sectionsá2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.
(d) Notwithstanding Sectioná2.2(b) above, no patent license is granted: (1)áfor any code that Contributor has deleted from the Contributor Version; (2)áfor infringements caused by: (i)áthird party modifications of Contributor Version, or (ii)áthe combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3)áunder Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.
You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.
You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.
You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipients rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.
You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.
Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any

subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.
4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a)árename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b)áotherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN AS IS BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as Participant) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sectionsá2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sectionsá6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF

SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTYS NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a commercial item, as that term is defined in 48áC.F.R.á2.101 (Oct. 1995), consisting of commercial computer software (as that term is defined at 48 C.F.R. á252.227-7014(a)(1)) and commercial computer software documentation as such terms are used in 48áC.F.R.á12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdictions conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)
The GlassFish code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa

Clara County, California.

1.92 jstl-api 1.2

1.92.1 Available under license :

```
/*
* DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.
*
* Copyright 1997-2008 Sun Microsystems, Inc. All rights reserved.
*
* The contents of this file are subject to the terms of either the GNU
* General Public License Version 2 only ("GPL") or the Common Development
* and Distribution License("CDDL") (collectively, the "License").  You
* may not use this file except in compliance with the License. You can obtain
* a copy of the License at https://glassfish.dev.java.net/public/CDDL+GPL.html
* or glassfish/bootstrap/legal/LICENSE.txt.  See the License for the specific
* language governing permissions and limitations under the License.
*
* When distributing the software, include this License Header Notice in each
* file and include the License file at glassfish/bootstrap/legal/LICENSE.txt.
* Sun designates this particular file as subject to the "Classpath" exception
* as provided by Sun in the GPL Version 2 section of the License file that
* accompanied this code.  If applicable, add the following below the License
* Header, with the fields enclosed by brackets [] replaced by your own
* identifying information: "Portions Copyrighted [year]
* [name of copyright owner]"
*
* Contributor(s):
*
* If you wish your version of this file to be governed by only the CDDL or
* only the GPL Version 2, indicate your decision by adding "[Contributor]
* elects to include this software in this distribution under the [CDDL or GPL
* Version 2] license."  If you don't indicate a single choice of license, a
* recipient has the option to distribute your version of this file under
* either the CDDL, the GPL Version 2 or to extend the choice of license to
* its licensees as provided above.  However, if you add GPL Version 2 code
* and therefore, elected the GPL Version 2 license, then the option applies
* only if the new code is made subject to such option by the copyright
* holder.
*
*
* This file incorporates work covered by the following copyright and
* permission notice:
*
* Copyright 2004 The Apache Software Foundation
*
* Licensed under the Apache License, Version 2.0 (the "License");
```

* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. "Contributor" means each individual or entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. "Covered Software" means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. "Executable" means the Covered Software in any form other than Source Code.

1.5. "Initial Developer" means the individual or entity that first makes Original Software available under this License.

1.6. "Larger Work" means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. "License" means this document.

1.8. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means the Source Code and Executable

form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. "Original Software" means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. "Source Code" means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer,

to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications

made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or

trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients" rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipient"s rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE

INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as "Participant") alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR

CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY"S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" (as that term is defined at 48 C.F.R. " 252.227-7014(a)(1)) and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdiction"s conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys" fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use,

distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module. An independent module is a module which is not derived from or based on this library. If you modify this library, you may extend this exception to your version of the library, but you are not obligated to do so. If you do not wish to do so, delete this exception statement from your version.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it

in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of

running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the

entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not

compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the

integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS

TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.
Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity
on behalf of whom a Contribution has been received by Licensor and
subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
copyright license to reproduce, prepare Derivative Works of,
publicly display, publicly perform, sublicense, and distribute the

Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside

or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,

and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.93 jta 1.1

1.93.1 Available under license :

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0
[OSI Approved License]

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. "Contributor" means each individual or entity that

creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. "Covered Software" means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. "Executable" means the Covered Software in any form other than Source Code.

1.5. "Initial Developer" means the individual or entity that first makes Original Software available under this License.

1.6. "Larger Work" means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. "License" means this document.

1.8. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. "Original Software" means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. "Source Code" means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients' rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred

by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipient's rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being

distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as "Participant") alleging that the Participant Software (meaning the

Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1

through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdiction's conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

1.94 Junit 4.10

1.94.1 Available under license :

Common Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC

LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
b) in the case of each subsequent Contributor:
i) changes to the Program, and
ii) additions to the Program;
where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.
b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.
c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual

property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.
d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.
3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and
b) its license agreement:
i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.
When the Program is made available in source code form:

a) it must be made available under this Agreement; and
b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.
BSD License

Copyright (c) 2000-2006, www.hamcrest.org
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of
conditions and the following disclaimer. Redistributions in binary form must reproduce
the above copyright notice, this list of conditions and the following disclaimer in
the documentation and/or other materials provided with the distribution.

Neither the name of Hamcrest nor the names of its contributors may be used to endorse
or promote products derived from this software without specific prior written
permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.95 junit_within-cglib 3.8.1

1.95.1 Available under license :

Common Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
b) in the case of each subsequent Contributor:
i) changes to the Program, and
ii) additions to the Program;
where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.
b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.
c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.
d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.
3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and
b) its license agreement:
i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange. When the Program is made available in source code form:

a) it must be made available under this Agreement; and
b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

1.96 keyczar 0.66

1.96.1 Available under license :

Apache License
Version 2.0, January 2008
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent

to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work,

excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any

risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!) The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.97 log4j 1.2.17

1.97.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object

form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a

file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright 1999-2005 The Apache Software Foundation

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache log4j
Copyright 2007 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).
Apache log4j
Copyright 2010 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).
<!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements. See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

-->

1.98 mail 1.4.7

1.98.1 Available under license :

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. Contributor. means each individual or entity that creates or contributes to the creation of Modifications.

1.2. Contributor Version. means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. Covered Software. means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. Executable. means the Covered Software in any form other than Source Code.

1.5. Initial Developer. means the individual or entity that first makes Original Software available under this License.

1.6. Larger Work. means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. License. means this document.

1.8. Licensable. means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications. means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. Original Software. means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. Patent Claims. means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. Source Code. means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. You. (or .Your.) means an individual or a legal entity exercising rights under, and complying with all of the

terms of, this License. For legal entities, .You. includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, .control. means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the

Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients. rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipient.s rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN .AS IS. BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as .Participant.) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software

against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY.S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a .commercial item,. as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of .commercial computer software. (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and .commercial computer software documentation. as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdiction.s conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys. fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)

The code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and

that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are

different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

One line to give the program's name and a brief idea of what it does.

Copyright (C)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

"CLASSPATH" EXCEPTION TO THE GPL VERSION 2

Certain source files distributed by Sun Microsystems, Inc. are subject to the following clarification and special exception to the GPL Version 2, but only where Sun has expressly included in the particular source file's header the words

"Sun designates this particular file as subject to the "Classpath" exception as provided by Sun in the License file that accompanied this code."

Linking this library statically or dynamically with other modules is making a combined work based on this library. Thus, the terms and conditions of the GNU General Public License Version 2 cover the whole combination.

As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module.? An independent module is a module which is not derived from or based on this library.? If you modify this library, you may extend this exception to your version of the library, but you are not obligated to do so.? If you do not wish to do so, delete this exception

statement from your version.

1.99 neethi 3.0 :3.0

1.99.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications

represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without

modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade

names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier

identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
Apache Neethi
Copyright 2004-2011 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

This product is tested with testcases developed at W3C under the license:
http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231
The source distribution of this product includes those testcases.

1.100 nekohtml 1.9.12

1.100.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the

direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of

this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and

wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor

has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

```
/*
* Copyright 2002-2008 Andy Clark
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
```

* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

1.101 org.apache.servicemix.specs.jsr311-api-1.0-1.2.0 1.2.0

1.101.1 Available under license :

Apache ServiceMix
Copyright 2007-2009 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation

source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable

(except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and

may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify,

defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.102 oro 2.0.8

1.102.1 Notifications :

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

1.102.2 Available under license :

```
/*
* $Id: MatchAction.java,v 1.7 2003/11/07 20:16:24 dfs Exp $
*
* ====================================================================
* The Apache Software License, Version 1.1
*
* Copyright (c) 2000 The Apache Software Foundation.  All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
```

```
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. The end-user documentation included with the redistribution,
*    if any, must include the following acknowledgment:
*       "This product includes software developed by the
*        Apache Software Foundation (http://www.apache.org/)."
*    Alternately, this acknowledgment may appear in the software itself,
*    if and wherever such third-party acknowledgments normally appear.
*
* 4. The names "Apache" and "Apache Software Foundation", "Jakarta-Oro"
*    must not be used to endorse or promote products derived from this
*    software without prior written permission. For written
*    permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache"
*    or "Jakarta-Oro", nor may "Apache" or "Jakarta-Oro" appear in their
*    name, without prior written permission of the Apache Software Foundation.
*
* THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* ====================================================================
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation.  For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*/
```

1.103 POI 3.9

1.103.1 Available under license :

Apache License, Version 2.0

Foundation
Projects
People
Get Involved
Download
Support Apache

Home » Licenses

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the

Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided

along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache POI
Copyright 2009 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

This product contains the DOM4J library (http://www.dom4j.org).
Copyright 2001-2005 (C) MetaStuff, Ltd. All Rights Reserved.

This product contains parts that were originally based on software from BEA.
Copyright (c) 2000-2003, BEA Systems, <http://www.bea.com/>.

This product contains W3C XML Schema documents. Copyright 2001-2003 (c)
World Wide Web Consortium (Massachusetts Institute of Technology, European
Research Consortium for Informatics and Mathematics, Keio University)

This product contains the Piccolo XML Parser for Java
(http://piccolo.sourceforge.net/). Copyright 2002 Yuval Oren.

This product contains the chunks_parse_cmds.tbl file from the vsdump program.
Copyright (C) 2006-2007 Valek Filippov (frob@df.ru)

1.104 quartz 1.6.1

1.104.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems,

and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and

limitations under the License.

```
=========================================================================
 ==  NOTICE file corresponding to the section 4 d of                ==
 ==  the Apache License, Version 2.0,                               ==
 ==  in this case for the Apache Camel distribution.                ==
 =========================================================================
```

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

Please read the different LICENSE files present in the licenses directory of
this distribution.

1.105 serializer 2.7.2

1.106 SLF4J LOG4J-12 Binding 1.5.8.PATCHED

1.106.1 Available under license :

```
/*
* Copyright (c) 2004-2008 QOS.ch
* All rights reserved.
*
* Permission is hereby granted, free  of charge, to any person obtaining
* a  copy  of this  software  and  associated  documentation files  (the
* "Software"), to  deal in  the Software without  restriction, including
* without limitation  the rights to  use, copy, modify,  merge, publish,
* distribute,  sublicense, and/or sell  copies of  the Software,  and to
* permit persons to whom the Software  is furnished to do so, subject to
* the following conditions:
*
* The  above  copyright  notice  and  this permission  notice  shall  be
* included in all copies or substantial portions of the Software.
*
* THE  SOFTWARE IS  PROVIDED  "AS  IS", WITHOUT  WARRANTY  OF ANY  KIND,
* EXPRESS OR  IMPLIED, INCLUDING  BUT NOT LIMITED  TO THE  WARRANTIES OF
* MERCHANTABILITY,  FITNESS  FOR  A  PARTICULAR  PURPOSE  AND
* NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
* LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
* OF CONTRACT, TORT OR OTHERWISE,  ARISING FROM, OUT OF OR IN CONNECTION
* WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
```

1.107 slf4j-api 1.6.1

1.107.1 Available under license :

```
/*
* Copyright (c) 2004-2007 QOS.ch
* All rights reserved.
*
* Permission is hereby granted, free  of charge, to any person obtaining
* a  copy  of this  software  and  associated  documentation files  (the
* "Software"), to  deal in  the Software without  restriction, including
* without limitation  the rights to  use, copy, modify,  merge, publish,
* distribute,  sublicense, and/or sell  copies of  the Software,  and to
* permit persons to whom the Software  is furnished to do so, subject to
* the following conditions:
*
* The  above  copyright  notice  and  this permission  notice  shall  be
* included in all copies or substantial portions of the Software.
*
* THE  SOFTWARE IS  PROVIDED  "AS  IS", WITHOUT  WARRANTY  OF ANY  KIND,
* EXPRESS OR  IMPLIED, INCLUDING  BUT NOT LIMITED  TO THE  WARRANTIES OF
* MERCHANTABILITY,  FITNESS  FOR  A  PARTICULAR  PURPOSE  AND
* NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
* LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
* OF CONTRACT, TORT OR OTHERWISE,  ARISING FROM, OUT OF OR IN CONNECTION
* WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
```

1.108 smtp 1.4.7

1.108.1 Available under license :

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0

1. Definitions.

1.1. Contributor. means each individual or entity that creates or contributes to the creation of Modifications.

1.2. Contributor Version. means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. Covered Software. means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. Executable. means the Covered Software in any form other than Source Code.

1.5. Initial Developer. means the individual or entity that first makes Original Software available under this

License.

1.6. Larger Work. means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. License. means this document.

1.8. Licensable. means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications. means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. Original Software. means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. Patent Claims. means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. Source Code. means (a) the common form of computer software code in which modifications are made and (b) associated documentation included in or with such code.

1.13. You. (or .Your.) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, .You. includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, .control. means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients. rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipient.s rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the

name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN .AS IS. BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as .Participant.) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY.S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF

INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a .commercial item,. as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of .commercial computer software. (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and .commercial computer software documentation. as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdiction.s conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys. fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

NOTICE PURSUANT TO SECTION 9 OF THE COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)

The code released under the CDDL shall be governed by the laws of the State of California (excluding conflict-of-law provisions). Any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California and the state courts of the State of California, with venue lying in Santa Clara County, California.

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307

USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term

"modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable

form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL,

SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

One line to give the program's name and a brief idea of what it does.

Copyright (C)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

"CLASSPATH" EXCEPTION TO THE GPL VERSION 2

Certain source files distributed by Sun Microsystems, Inc. are subject to the following clarification and special exception to the GPL Version 2, but only where Sun has expressly included in the particular source file's header the words

"Sun designates this particular file as subject to the "Classpath" exception as provided by Sun in the License file that accompanied this code."

Linking this library statically or dynamically with other modules is making a combined work based on this library. Thus, the terms and conditions of the GNU General Public License Version 2 cover the whole combination.

As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module.? An independent module is a module which is not derived from or based on this library.? If you modify this library, you may extend this exception to your version of the library, but you are not obligated to do so.? If you do not wish to do so, delete this exception statement from your version.

1.109 SNMP4J 1.11.2

1.109.1 Available under license :

Apache License

Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not

pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special,

incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

```
================================================================================
 ==  NOTICE file corresponding to the section 4 d of                ==
 ==  the Apache License, Version 2.0,                               ==
 ==  in this case for the SNMP4J distribution.                       ==
```

```
=========================================================================

This product includes software developed by
SNMP4J.org (http://www.snmp4j.org/).

Please read the different LICENSE files present in the root directory of
this distribution.

The names "SNMP4J" and  "Apache Software Foundation"  must not be used to
endorse  or promote  products derived  from this  software without prior
written permission. For written permission, please contact
info@snmp4j.org (SNMP4J) or apache@apache.org.
/*
* The Apache Software License, Version 1.1
*
*
* Copyright (c) 1999-2002 The Apache Software Foundation.  All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. The end-user documentation included with the redistribution,
*    if any, must include the following acknowledgment:
*       "This product includes software developed by the
*        Apache Software Foundation (http://www.apache.org/)."
*    Alternately, this acknowledgment may appear in the software itself,
*    if and wherever such third-party acknowledgments normally appear.
*
* 4. The names "Xerces" and "Apache Software Foundation" must
*    not be used to endorse or promote products derived from this
*    software without prior written permission. For written
*    permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache",
*    nor may "Apache" appear in their name, without prior written
*    permission of the Apache Software Foundation.
*
* THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
```

* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* ==
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation and was
* originally based on software copyright (c) 1999, International
* Business Machines, Inc., http://www.ibm.com. For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*/

1.110 spring-aop 3.1.0.RELEASE

1.110.1 Available under license :

/*
* Copyright 2002-2006 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to

You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and

assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work
To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.111 spring-asm 3.1.0.RELEASE

1.111.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of

the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works

that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A

PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

ASM: a very small and fast Java bytecode manipulation framework
Copyright (c) 2000-2005 INRIA, France Telecom
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holders nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.

1.112 spring-aspects 3.1.0.RELEASE

1.112.1 Available under license :

```
/*
* Copyright 2002-2011 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
```

* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the

Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any

additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work
To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.113 spring-beans 3.1.0.RELEASE

1.113.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object

form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a

file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

```
/*
* Copyright 2002-2009 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
```

1.114 spring-context 3.1.0.RELEASE

1.114.1 Available under license :

```
/*
* Copyright 2002-2009 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
```

* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the

Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any

additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work
To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.115 spring-context-support 3.1.0.RELEASE

1.115.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object

form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a

file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

```
/*
* Copyright 2002-2010 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
# Additional file types adapted from
# http://www.utoronto.ca/webdocs/HTMLdocs/Book/Book-3ed/appb/mimetype.html
# kindly re-licensed to Apache Software License 2.0 by Ian Graham.
```

1.116 spring-core 3.1.0.RELEASE

1.116.1 Available under license :

```
/*
* Copyright 2002-2006 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
```

*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright

owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work
To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.117 spring-expression 3.1.0.RELEASE

1.117.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work

(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses

granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]"

replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

```
/*
* Copyright 2002-2009 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
```

1.118 spring-jdbc 3.1.0.RELEASE

1.118.1 Available under license :

```
/*
* Copyright 2002-2006 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
```

*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright

owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work
To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.119 spring-jms 3.1.0.RELEASE

1.119.1 Available under license :

/*
* Copyright 2002-2007 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend,

and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work
To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.120 spring-orm 3.1.0.RELEASE

1.120.1 Available under license :

/*
* Copyright 2002-2006 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to

You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work
To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.121 spring-oxm-3.1.0.RELEASE

3.1.0.RELEASE

1.122 spring-test 3.1.0.RELEASE

1.122.1 Available under license :

/*
* Copyright 2002-2008 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf

and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work
To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

1.123 spring-tx 3.1.0.RELEASE

1.124 spring-web 3.1.0.RELEASE

1.124.1 Available under license :

```
/*
* Copyright 2002-2011 the original author or authors.
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

* See the License for the specific language governing permissions and
* limitations under the License.
*/
Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously

marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of

any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work
To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

1.125 spring-webmvc 3.1.0.RELEASE

1.126 spring-xml 1.5.2

1.127 StAX API 1.0.1

1.127.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and
You must cause any modified files to carry prominent notices stating that You changed the files; and
You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such

third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within

third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
/*
**
** Licensed to the Apache Software Foundation (ASF) under one
** or more contributor license agreements. See the NOTICE file
** distributed with this work for additional information
** regarding copyright ownership. The ASF licenses this file
** to you under the Apache License, Version 2.0 (the
** "License"); you may not use this file except in compliance
** with the License. You may obtain a copy of the License at
**
** http://www.apache.org/licenses/LICENSE-2.0
**
** Unless required by applicable law or agreed to in writing,
** software distributed under the License is distributed on an
** "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
** KIND, either express or implied. See the License for the
** specific language governing permissions and limitations
** under the License.
*/

1.128 super-csv 2.3.1

1.129 velocity 1.7

1.129.1 Available under license :

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of

the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works

that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A

PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache Velocity

Copyright (C) 2000-2007 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

1.130 wsdl4j 1.6.2

1.130.1 Available under license :

Common Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
b) in the case of each subsequent Contributor:
i) changes to the Program, and
ii) additions to the Program;
where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.
b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.
c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.
d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.
3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and
b) its license agreement:
i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.
When the Program is made available in source code form:

a) it must be made available under this Agreement; and
b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF

THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

1.131 Xalan 2.7.1

1.131.1 Available under license :

?<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications

represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without

modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade

names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

>>

The license above applies to this Apache Xalan release of:
Xalan-Java 2 - XSLT Processor
Xalan-Java 2 - Serializer

The license above also applies to the jar files
xalan.jar and xsltc.jar - Xalan-Java 2 - XSLT Processor from
Source: http://xalan.apache.org/

The license above also applies to the jar file
serializer.jar - Xalan-Java 2 - Serializer
Source: http://xalan.apache.org/
Used by: Xalan-Java 2 and Xerces-Java 2

The license above also applies to the jar file
xercesImpl.jar - Xerces-Java 2 XML Parser.
Source: http://xerces.apache.org/
Used by: Xalan-Java 2

The license above also applies to the jar file
xml-apis.jar - Xerces-Java 2 XML Parser.
Source: http://xerces.apache.org/
Used by: Xalan-Java 2 and release copy of Xerces-Java 2

The following license applies to the included files:
tools/ant.jar
tools/antRun
tools/antRun.bat
Source: http://ant.apache.org/
Used By: Xalan's build process: java/build.xml and test/build.xml

```
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
/*
* ============================================================================
*                The Apache Software License, Version 1.1
* ============================================================================
*
*    Copyright (C) 1999 The Apache Software Foundation. All rights reserved.
*
* Redistribution and use in source and binary forms, with or without modifica-
* tion, are permitted provided that the following conditions are met:
*
* 1. Redistributions of  source code must  retain the above copyright  notice,
*    this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice,
*    this list of conditions and the following disclaimer in the documentation
*    and/or other materials provided with the distribution.
*
* 3. The end-user documentation included with the redistribution, if any, must
```

>>

The following license, Apache Software License, Version 1.1,
applies to the included BCEL.jar from Apache Jakarta
(Byte Code Engineering Library).
Source: http://jakarta.apache.org/bcel
Used By: XSLTC component of xml-xalan/java

The following license, Apache Software License, Version 1.1,
also applies to the included regexp.jar,
jakarta-regexp-1.2.jar from Apache Jakarta.

Source: http://jakarta.apache.org/regexp
Used By: BCEL.jar which is used by XSLTC component of xml-xalan/java

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
/*
*
* Copyright (c) 2001 The Apache Software Foundation. All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*   notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*   notice, this list of conditions and the following disclaimer in
*   the documentation and/or other materials provided with the
*   distribution.
*
* 3. The end-user documentation included with the redistribution,
*   if any, must include the following acknowledgment:
*   "This product includes software developed by the
*   Apache Software Foundation (http://www.apache.org/)."
*   Alternately, this acknowledgment may appear in the software itself,
*   if and wherever such third-party acknowledgments normally appear.
*
* 4. The names "Apache" and "Apache Software Foundation" and
*   "Apache BCEL" must not be used to endorse or promote products
*   derived from this software without prior written permission. For
*   written permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache",
*   "Apache BCEL", nor may "Apache" appear in their name, without
*   prior written permission of the Apache Software Foundation.
*
* THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF

* SUCH DAMAGE.
* ==
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation. For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*/
>>

The following license applies to the DOM documentation
for the org.w3c.dom.* packages:

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
W3C? DOCUMENT LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231
Public documents on the W3C site are provided by the copyright holders
under the following license. By using and/or copying this document,
or the W3C document from which this statement is linked, you (the licensee)
agree that you have read, understood, and will comply with the following
terms and conditions:

Permission to copy, and distribute the contents of this document, or the
W3C document from which this statement is linked, in any medium for any
purpose and without fee or royalty is hereby granted, provided that you include
the following on ALL copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it
  doesn't exist, a notice (hypertext is preferred, but a textual representation
   is permitted) of the form: "Copyright) [$date-of-document] World Wide Web
   Consortium, (Massachusetts Institute of Technology, European Research
   Consortium for Informatics and Mathematics, Keio University). All Rights
   Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231"
3. If it exists, the STATUS of the W3C document.

When space permits, inclusion of the full text of this NOTICE should be provided.
We request that authorship attribution be provided in any software, documents,
or other items or products that you create pursuant to the implementation of the
contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant
to this license. However, if additional requirements (documented in the Copyright FAQ)
are satisfied, the right to create modifications or derivatives is sometimes granted
by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS
OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE;
THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE
IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS,
COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE
PERFORMANCE
OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising
or publicity pertaining to this document or its contents without specific,
written prior permission. Title to copyright in this document will at all
times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002.
This version removes the copyright ownership notice such that this license
can be used with materials other than those owned by the W3C, moves information
on style sheets, DTDs, and schemas to the Copyright FAQ, reflects that ERCIM
is now a host of the W3C, includes references to this specific dated version
of the license, and removes the ambiguous grant of "use". See the older
formulation for the policy prior to this date. Please see our Copyright FAQ for
common questions about using materials from our site, such as the translating
or annotating specifications. Other questions about this notice can be directed
to site-policy@w3.org.

Joseph Reagle <mailto:site-policy@w3.org
Last revised by Reagle $Date: 2005-07-19 12:33:09 -0400 (Tue, 19 Jul 2005) $
>>>

The following license applies to the DOM software,
for the org.w3c.dom.* packages in jar file xml-apis.jar:

This version removes the copyright ownership notice such that this license can
be used with materials other than those owned by the W3C, reflects that ERCIM
is now a host of the W3C, includes references to this specific dated version
of the license, and removes the ambiguous grant of "use". Otherwise, this
version is the same as the previous version and is written so as to preserve
the Free Software Foundation's assessment of GPL compatibility and OSI's
certification under the Open Source Definition. Please see our Copyright FAQ
for common questions about using materials from our site, including specific
terms and conditions for packages like libwww, Amaya, and Jigsaw. Other
questions about this notice can be directed to site-policy@w3.org.

Joseph Reagle <mailto:site-policy@w3.org
Last revised by Reagle $Date: 2005-07-19 12:33:09 -0400 (Tue, 19 Jul 2005) $
>>

The following license applies to the SAX software,
for the org.xml.sax.* packages in jar file xml-apis.jar:

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
This module, both source code and documentation, is in the Public Domain,
and comes with NO WARRANTY. See http://www.saxproject.org for further information.
>>>

The following license applies to the jar file
java_cup.jar - LALR Parser Generator for Java(TM).
Source: http://www.cs.princeton.edu/~appel/modern/java/CUP
Used By: XSLTC component of xml-xalan/java

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CUP Parser Generator Copyright Notice, License, and Disclaimer

Copyright 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided
that the above copyright notice appear in all copies and that both
the copyright notice and this permission notice and warranty disclaimer
appear in supporting documentation, and that the names of the authors
or their employers not be used in advertising or publicity pertaining
to distribution of the software without specific, written prior permission.

The authors and their employers disclaim all warranties with regard to
this software, including all implied warranties of merchantability
and fitness. In no event shall the authors or their employers be liable
for any special, indirect or consequential damages or any damages
whatsoever resulting from loss of use, data or profits, whether in an action
of contract, negligence or other tortious action, arising out of or
in connection with the use or performance of this software.
>>>

The following license applies to the jar file runtime.jar - Component
of JavaCup: LALR Parser Generator for Java(TM).
Source: http://www.cs.princeton.edu/~appel/modern/java/CUP
Used By: XSLTC component of xml-xalan/java

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CUP Parser Generator Copyright Notice, License, and Disclaimer
(runtime.jar component)

Copyright 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided
that the above copyright notice appear in all copies and that both
the copyright notice and this permission notice and warranty disclaimer
appear in supporting documentation, and that the names of the authors
or their employers not be used in advertising or publicity pertaining
to distribution of the software without specific, written prior permission.

The authors and their employers disclaim all warranties with regard to
this software, including all implied warranties of merchantability
and fitness. In no event shall the authors or their employers be liable
for any special, indirect or consequential damages or any damages
whatsoever resulting from loss of use, data or profits, whether in an action

of contract, negligence or other tortious action, arising out of or
in connection with the use or performance of this software.
>>

The following license applies to the JLEX jar file
JLex.jar - A Lexical Analyzer Generator for Java(TM).
Source: http://www.cs.princeton.edu/~appel/modern/java/JLex
Used By: XSLTC component of xml-xalan/java

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
JLEX COPYRIGHT NOTICE, LICENSE AND DISCLAIMER.

Copyright 1996-2000 by Elliot Joel Berk and C. Scott Ananian

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and
that both the copyright notice and this permission notice and
warranty disclaimer appear in supporting documentation, and that the
name of the authors or their employers not be used in advertising or
publicity pertaining to distribution of the software without specific,
written prior permission.

The authors and their employers disclaim all warranties with regard
to this software, including all implied warranties of merchantability and
fitness. In no event shall the authors or their employers be liable for any
special, indirect or consequential damages or any damages whatsoever resulting
from loss of use, data or profits, whether in an action of contract,
negligence or other tortious action, arising out of or in connection
with the use or performance of this software.

Java is a trademark of Sun Microsystems, Inc. References to the Java
programming language in relation to JLex are not meant to imply that
Sun endorses this product.
>>

The following license applies to the jar file
stylebook-1.0-b3_xalan-2.jar - Tool for generating Xalan documentation.
Integrated with Xalan-Java 2 and Xerces 2.
Source: http://svn.apache.org/viewvc/xml/stylebook/
Used by: Xalan-Java 2, Xalan-C++

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
/*
* The Apache Software License, Version 1.1
*
*
* Copyright (c) 1999 The Apache Software Foundation. All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in
* the documentation and/or other materials provided with the
* distribution.
*
* 3. The end-user documentation included with the redistribution,
* if any, must include the following acknowledgment:
* "This product includes software developed by the
* Apache Software Foundation (http://www.apache.org/)."
* Alternately, this acknowledgment may appear in the software itself,
* if and wherever such third-party acknowledgments normally appear.
*
* 4. The names "Xalan", "Xerces", and "Apache Software Foundation" must
* not be used to endorse or promote products derived from this
* software without prior written permission. For written
* permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache",
* nor may "Apache" appear in their name, without prior written
* permission of the Apache Software Foundation.
*
* THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR

* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* ===
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation and was
* originally based on software copyright (c) 1999, International
* Business Machines, Inc., http://www.apache.org. For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*/
>>>
===
 == NOTICE file corresponding to section 4(d) of the Apache License,   ==
 == Version 2.0, in this case for the Apache Xalan Java distribution. ==
 ===

 Apache Xalan (Xalan XSLT processor)
 Copyright 1999-2006 The Apache Software Foundation

 Apache Xalan (Xalan serializer)
 Copyright 1999-2006 The Apache Software Foundation

 This product includes software developed at
 The Apache Software Foundation (http://www.apache.org/).

 ===
 Portions of this software was originally based on the following:
   - software copyright (c) 1999-2002, Lotus Development Corporation.,
     http://www.lotus.com.
   - software copyright (c) 2001-2002, Sun Microsystems.,
     http://www.sun.com.
   - software copyright (c) 2003, IBM Corporation.,
     http://www.ibm.com.

 ===
 The binary distribution package (ie. jars, samples and documentation) of
 this product includes software developed by the following:

   - The Apache Software Foundation
       - Xerces Java - see LICENSE.txt
       - JAXP 1.3 APIs - see LICENSE.txt

- Bytecode Engineering Library - see LICENSE.txt
- Regular Expression - see LICENSE.txt

- Scott Hudson, Frank Flannery, C. Scott Ananian
  - CUP Parser Generator runtime (javacup\runtime) - see LICENSE.txt

==========

The source distribution package (ie. all source and tools required to build
Xalan Java) of this product includes software developed by the following:

- The Apache Software Foundation
  - Xerces Java - see LICENSE.txt
  - JAXP 1.3 APIs - see LICENSE.txt
  - Bytecode Engineering Library - see LICENSE.txt
  - Regular Expression - see LICENSE.txt
  - Ant - see LICENSE.txt
  - Stylebook doc tool - see LICENSE.txt

- Elliot Joel Berk and C. Scott Ananian
  - Lexical Analyzer Generator (JLex) - see LICENSE.txt

==========

Apache Xerces Java
Copyright 1999-2006 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Portions of Apache Xerces Java in xercesImpl.jar and xml-apis.jar
were originally based on the following:
- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
- voluntary contributions made by Paul Eng on behalf of the
  Apache Software Foundation that were originally developed at iClick, Inc.,
  software copyright (c) 1999.

==========

Apache xml-commons xml-apis (redistribution of xml-apis.jar)

Apache XML Commons
Copyright 2001-2003,2006 The Apache Software Foundation.

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Portions of this software were originally based on the following:
- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.

- software copyright (c) 2000 World Wide Web Consortium, http://www.w3.org

1.132 Xalan 2.7.2

1.132.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the

editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the

Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the

same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache Geronimo
Copyright 2003-2006 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

Apache OpenEJB
Copyright 1999-2009 The Apache OpenEJB development community

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).
xml-commons/java/external/LICENSE.dom-software.txt $Id: LICENSE.dom-software.txt,v 1.2 2005/06/03 22:49:13 mrglavas Exp $

This license came from: http://www.w3.org/Consortium/Legal/copyright-software-20021231

W3C SOFTWARE NOTICE AND LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

This work (and included software, documentation such as READMEs, or other
related items) is being provided by the copyright holders under the following
license. By obtaining, using and/or copying this work, you (the licensee) agree
that you have read, understood, and will comply with the following terms and
conditions.

Permission to copy, modify, and distribute this software and its documentation,
with or without modification, for any purpose and without fee or royalty is
hereby granted, provided that you include the following on ALL copies of the
software and documentation or portions thereof, including modifications:

1. The full text of this NOTICE in a location viewable to users of the
   redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices, or terms
   and conditions. If none exist, the W3C Software Short Notice should be
   included (hypertext is preferred, text is permitted) within the body
   of any redistributed or derivative code.
3. Notice of any changes or modifications to the files, including the date
   changes were made. (We recommend you provide URIs to the location from
   which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002. This version removes the copyright ownership notice such that this license can be used with materials other than those owned by the W3C, reflects that ERCIM is now a host of the W3C, includes references to this specific dated version of the license, and removes the ambiguous grant of "use". Otherwise, this version is the same as the previous version and is written so as to preserve the Free Software Foundation's assessment of GPL compatibility and OSI's certification under the Open Source Definition. Please see our Copyright FAQ for common questions about using materials from our site, including specific terms and conditions for packages like libwww, Amaya, and Jigsaw. Other questions about this notice can be directed to site-policy@w3.org.

Joseph Reagle <site-policy@w3.org>

Last revised by Reagle $Date: 2005/06/03 22:49:13 $

```
 =========================================================================
 ==  NOTICE file corresponding to section 4(d) of the Apache License,   ==
 ==  Version 2.0, in this case for the Apache xml-commons xml-apis      ==
 ==  distribution.                                                      ==
 =========================================================================

  This product includes software developed by
```

The Apache Software Foundation (http://www.apache.org/).

Portions of this software were originally based on the following:
- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
- software copyright (c) 2000 World Wide Web Consortium, http://www.w3.org

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object

form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

(a) You must give any other recipients of the Work or
Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
that You distribute, all copyright, patent, trademark, and
attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

>>

The license above applies to this Apache Xalan release of:
Xalan-Java 2 - XSLT Processor
Xalan-Java 2 - Serializer

The license above also applies to the jar files
xalan.jar and xsltc.jar - Xalan-Java 2 - XSLT Processor from

Source: http://xalan.apache.org/

The license above also applies to the jar file
serializer.jar - Xalan-Java 2 - Serializer
Source: http://xalan.apache.org/
Used by: Xalan-Java 2 and Xerces-Java 2

The license above also applies to the jar file
xercesImpl.jar - Xerces-Java 2 XML Parser.
Source: http://xerces.apache.org/
Used by: Xalan-Java 2

The license above also applies to the jar file
xml-apis.jar - Xerces-Java 2 XML Parser.
Source: http://xerces.apache.org/
Used by: Xalan-Java 2 and release copy of Xerces-Java 2

The following license applies to the included files:
tools/ant.jar
tools/antRun
tools/antRun.bat
Source: http://ant.apache.org/
Used By: Xalan's build process: java/build.xml and test/build.xml

```
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
/*
* ============================================================================
*                The Apache Software License, Version 1.1
* ============================================================================
*
*    Copyright (C) 1999 The Apache Software Foundation. All rights reserved.
*
* Redistribution and use in source and binary forms, with or without modifica-
* tion, are permitted provided that the following conditions are met:
*
* 1. Redistributions of  source code must  retain the above copyright  notice,
*    this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice,
*    this list of conditions and the following disclaimer in the documentation
*    and/or other materials provided with the distribution.
```

```
*
* 3. The end-user documentation included with the redistribution, if any, must
*    include  the following  acknowledgment:  "This product includes  software
*    developed  by the  Apache Software Foundation  (http://www.apache.org/)."
*    Alternately, this  acknowledgment may  appear in the software itself,  if
*    and wherever such third-party acknowledgments normally appear.
*
* 4. The names "Ant" and  "Apache Software Foundation"  must not be used to
*    endorse  or promote  products derived  from this  software without  prior
*    written permission. For written permission, please contact
*    apache@apache.org.
*
* 5. Products  derived from this software may not  be called "Apache", nor may
*    "Apache" appear  in their name,  without prior written permission  of the
*    Apache Software Foundation.
*
* THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES,
* INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
* FITNESS  FOR A PARTICULAR  PURPOSE ARE  DISCLAIMED.  IN NO  EVENT SHALL  THE
* APACHE SOFTWARE  FOUNDATION  OR ITS CONTRIBUTORS  BE LIABLE FOR  ANY DIRECT,
* INDIRECT, INCIDENTAL, SPECIAL,  EXEMPLARY, OR CONSEQUENTIAL  DAMAGES (INCLU-
* DING, BUT NOT LIMITED TO, PROCUREMENT  OF SUBSTITUTE GOODS OR SERVICES; LOSS
* OF USE, DATA, OR  PROFITS; OR BUSINESS  INTERRUPTION)  HOWEVER CAUSED AND ON
* ANY  THEORY OF LIABILITY,  WHETHER  IN CONTRACT,  STRICT LIABILITY,  OR TORT
* (INCLUDING  NEGLIGENCE OR  OTHERWISE) ARISING IN  ANY WAY OUT OF THE  USE OF
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
* This software  consists of voluntary contributions made  by many individuals
* on behalf of the  Apache Software Foundation.  For more  information  on the
* Apache Software Foundation, please see <http://www.apache.org/>.
*
*/
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

The following license, Apache Software License, Version 1.1,
applies to the included BCEL.jar from Apache Jakarta
(Byte Code Engineering Library).
Source: http://jakarta.apache.org/bcel
Used By: XSLTC component of xml-xalan/java

The following license, Apache Software License, Version 1.1,

also applies to the included regexp.jar,
jakarta-regexp-1.2.jar from Apache Jakarta.
Source: http://jakarta.apache.org/regexp
Used By: BCEL.jar which is used by XSLTC component of xml-xalan/java

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
/*
*
* Copyright (c) 2001 The Apache Software Foundation. All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
* notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
* notice, this list of conditions and the following disclaimer in
* the documentation and/or other materials provided with the
* distribution.
*
* 3. The end-user documentation included with the redistribution,
* if any, must include the following acknowledgment:
* "This product includes software developed by the
* Apache Software Foundation (http://www.apache.org/)."
* Alternately, this acknowledgment may appear in the software itself,
* if and wherever such third-party acknowledgments normally appear.
*
* 4. The names "Apache" and "Apache Software Foundation" and
* "Apache BCEL" must not be used to endorse or promote products
* derived from this software without prior written permission. For
* written permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache",
* "Apache BCEL", nor may "Apache" appear in their name, without
* prior written permission of the Apache Software Foundation.
*
* THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,

* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* ==
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation. For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*/
>>>

The following license applies to the DOM documentation
for the org.w3c.dom.* packages:

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
W3C DOCUMENT LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231
Public documents on the W3C site are provided by the copyright holders
under the following license. By using and/or copying this document,
or the W3C document from which this statement is linked, you (the licensee)
agree that you have read, understood, and will comply with the following
terms and conditions:

Permission to copy, and distribute the contents of this document, or the
W3C document from which this statement is linked, in any medium for any
purpose and without fee or royalty is hereby granted, provided that you include
the following on ALL copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it
  doesn't exist, a notice (hypertext is preferred, but a textual representation
   is permitted) of the form: "Copyright [$date-of-document] World Wide Web
   Consortium, (Massachusetts Institute of Technology, European Research
   Consortium for Informatics and Mathematics, Keio University). All Rights
   Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231"
3. If it exists, the STATUS of the W3C document.

When space permits, inclusion of the full text of this NOTICE should be provided.
We request that authorship attribution be provided in any software, documents,
or other items or products that you create pursuant to the implementation of the

contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant
to this license. However, if additional requirements (documented in the Copyright FAQ)
are satisfied, the right to create modifications or derivatives is sometimes granted
by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS
OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE;
THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE
IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS,
COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE
PERFORMANCE
OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising
or publicity pertaining to this document or its contents without specific,
written prior permission. Title to copyright in this document will at all
times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002.
This version removes the copyright ownership notice such that this license
can be used with materials other than those owned by the W3C, moves information
on style sheets, DTDs, and schemas to the Copyright FAQ, reflects that ERCIM
is now a host of the W3C, includes references to this specific dated version
of the license, and removes the ambiguous grant of "use". See the older
formulation for the policy prior to this date. Please see our Copyright FAQ for
common questions about using materials from our site, such as the translating
or annotating specifications. Other questions about this notice can be directed
to site-policy@w3.org.

Joseph Reagle <mailto:site-policy@w3.org
Last revised by Reagle $Date: 2005-07-19 12:33:09 -0400 (Tue, 19 Jul 2005) $
>>>

The following license applies to the DOM software,
for the org.w3c.dom.* packages in jar file xml-apis.jar:

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
W3C SOFTWARE NOTICE AND LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
This work (and included software, documentation such as READMEs,
or other related items) is being provided by the copyright holders
under the following license. By obtaining, using and/or copying this
work, you (the licensee) agree that you have read, understood, and will
comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its
documentation, with or without modification, for any purpose and
without fee or royalty is hereby granted, provided that you include
the following on ALL copies of the software and documentation or
portions thereof, including modifications:

1. The full text of this NOTICE in a location viewable to users of the
  redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices,
  or terms and conditions. If none exist, the W3C Software Short Notice
  should be included (hypertext is preferred, text is permitted) within
  the body of any redistributed or derivative code.
3. Notice of any changes or modifications to the files, including the
  date changes were made. (We recommend you provide URIs to the location
  from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS
MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT
NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR
PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE
ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising
or publicity pertaining to the software without specific, written prior
permission. Title to copyright in this software and any associated documentation
will at all times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002.
This version removes the copyright ownership notice such that this license can
be used with materials other than those owned by the W3C, reflects that ERCIM
is now a host of the W3C, includes references to this specific dated version
of the license, and removes the ambiguous grant of "use". Otherwise, this
version is the same as the previous version and is written so as to preserve
the Free Software Foundation's assessment of GPL compatibility and OSI's
certification under the Open Source Definition. Please see our Copyright FAQ
for common questions about using materials from our site, including specific
terms and conditions for packages like libwww, Amaya, and Jigsaw. Other
questions about this notice can be directed to site-policy@w3.org.

Joseph Reagle <mailto:site-policy@w3.org
Last revised by Reagle $Date: 2005-07-19 12:33:09 -0400 (Tue, 19 Jul 2005) $
>>>

The following license applies to the SAX software,
for the org.xml.sax.* packages in jar file xml-apis.jar:

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
This module, both source code and documentation, is in the Public Domain,
and comes with NO WARRANTY. See http://www.saxproject.org for further information.
>>>

The following license applies to the jar file
java_cup.jar - LALR Parser Generator for Java(TM).
Source: http://www.cs.princeton.edu/~appel/modern/java/CUP
Used By: XSLTC component of xml-xalan/java

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CUP Parser Generator Copyright Notice, License, and Disclaimer

Copyright 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided
that the above copyright notice appear in all copies and that both
the copyright notice and this permission notice and warranty disclaimer
appear in supporting documentation, and that the names of the authors
or their employers not be used in advertising or publicity pertaining
to distribution of the software without specific, written prior permission.

The authors and their employers disclaim all warranties with regard to
this software, including all implied warranties of merchantability
and fitness. In no event shall the authors or their employers be liable
for any special, indirect or consequential damages or any damages
whatsoever resulting from loss of use, data or profits, whether in an action
of contract, negligence or other tortious action, arising out of or
in connection with the use or performance of this software.
>>

The following license applies to the jar file runtime.jar - Component
of JavaCup: LALR Parser Generator for Java(TM).
Source: http://www.cs.princeton.edu/~appel/modern/java/CUP
Used By: XSLTC component of xml-xalan/java

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CUP Parser Generator Copyright Notice, License, and Disclaimer
(runtime.jar component)

Copyright 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided
that the above copyright notice appear in all copies and that both
the copyright notice and this permission notice and warranty disclaimer
appear in supporting documentation, and that the names of the authors
or their employers not be used in advertising or publicity pertaining
to distribution of the software without specific, written prior permission.

The authors and their employers disclaim all warranties with regard to
this software, including all implied warranties of merchantability
and fitness. In no event shall the authors or their employers be liable
for any special, indirect or consequential damages or any damages

whatsoever resulting from loss of use, data or profits, whether in an action
of contract, negligence or other tortious action, arising out of or
in connection with the use or performance of this software.
>>

The following license applies to the JLEX jar file
JLex.jar - A Lexical Analyzer Generator for Java(TM).
Source: http://www.cs.princeton.edu/~appel/modern/java/JLex
Used By: XSLTC component of xml-xalan/java

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
JLEX COPYRIGHT NOTICE, LICENSE AND DISCLAIMER.

Copyright 1996-2000 by Elliot Joel Berk and C. Scott Ananian

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and
that both the copyright notice and this permission notice and
warranty disclaimer appear in supporting documentation, and that the
name of the authors or their employers not be used in advertising or
publicity pertaining to distribution of the software without specific,
written prior permission.

The authors and their employers disclaim all warranties with regard
to this software, including all implied warranties of merchantability and
fitness. In no event shall the authors or their employers be liable for any
special, indirect or consequential damages or any damages whatsoever resulting
from loss of use, data or profits, whether in an action of contract,
negligence or other tortious action, arising out of or in connection
with the use or performance of this software.

Java is a trademark of Sun Microsystems, Inc. References to the Java
programming language in relation to JLex are not meant to imply that
Sun endorses this product.
>>

The following license applies to the jar file
stylebook-1.0-b3_xalan-2.jar - Tool for generating Xalan documentation.
Integrated with Xalan-Java 2 and Xerces 2.
Source: http://svn.apache.org/viewvc/xml/stylebook/
Used by: Xalan-Java 2, Xalan-C++

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

```
/*
* The Apache Software License, Version 1.1
*
*
* Copyright (c) 1999 The Apache Software Foundation.  All rights
* reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. The end-user documentation included with the redistribution,
*    if any, must include the following acknowledgment:
*       "This product includes software developed by the
*        Apache Software Foundation (http://www.apache.org/)."
*    Alternately, this acknowledgment may appear in the software itself,
*    if and wherever such third-party acknowledgments normally appear.
*
* 4. The names "Xalan", "Xerces", and "Apache Software Foundation" must
*    not be used to endorse or promote products derived from this
*    software without prior written permission. For written
*    permission, please contact apache@apache.org.
*
* 5. Products derived from this software may not be called "Apache",
*    nor may "Apache" appear in their name, without prior written
*    permission of the Apache Software Foundation.
*
* THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED
* WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
```

* DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* ==
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation and was
* originally based on software copyright (c) 1999, International
* Business Machines, Inc., http://www.apache.org. For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*/
>>
===
== NOTICE file corresponding to section 4(d) of the Apache License,   ==
== Version 2.0, in this case for the Apache Xalan Java distribution. ==
===

Apache Xalan (Xalan XSLT processor)
Copyright 1999-2014 The Apache Software Foundation

Apache Xalan (Xalan serializer)
Copyright 1999-2012 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

===
Portions of this software was originally based on the following:
- software copyright (c) 1999-2002, Lotus Development Corporation., http://www.lotus.com.
- software copyright (c) 2001-2002, Sun Microsystems., http://www.sun.com.
- software copyright (c) 2003, IBM Corporation., http://www.ibm.com.

===
The binary distribution package (ie. jars, samples and documentation) of
this product includes software developed by the following:

- The Apache Software Foundation
    - Xerces Java - see LICENSE.txt

- JAXP 1.3 APIs - see LICENSE.txt
- Bytecode Engineering Library - see LICENSE.txt
- Regular Expression - see LICENSE.txt

- Scott Hudson, Frank Flannery, C. Scott Ananian
- CUP Parser Generator runtime (javacup\runtime) - see LICENSE.txt

==========

The source distribution package (ie. all source and tools required to build
Xalan Java) of this product includes software developed by the following:

- The Apache Software Foundation
- Xerces Java - see LICENSE.txt
- JAXP 1.3 APIs - see LICENSE.txt
- Bytecode Engineering Library - see LICENSE.txt
- Regular Expression - see LICENSE.txt
- Ant - see LICENSE.txt
- Stylebook doc tool - see LICENSE.txt

- Elliot Joel Berk and C. Scott Ananian
- Lexical Analyzer Generator (JLex) - see LICENSE.txt

==========

Apache Xerces Java
Copyright 1999-2006 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Portions of Apache Xerces Java in xercesImpl.jar and xml-apis.jar
were originally based on the following:
- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
- voluntary contributions made by Paul Eng on behalf of the
Apache Software Foundation that were originally developed at iClick, Inc.,
software copyright (c) 1999.

==========

Apache xml-commons xml-apis (redistribution of xml-apis.jar)

Apache XML Commons
Copyright 2001-2003,2006 The Apache Software Foundation.

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Portions of this software were originally based on the following:
- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.

- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
- software copyright (c) 2000 World Wide Web Consortium, http://www.w3.org

SUN PUBLIC LICENSE Version 1.0

1. Definitions.

1.0.1. "Commercial Use" means distribution or otherwise making the Covered Code available to a third party.

1.1. "Contributor" means each entity that creates or contributes to the creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.

1.3. "Covered Code" means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof and corresponding documentation released with the source code.

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. "License" means this document.

1.8.1. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. "Original Code" means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.10.1. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.11. "Source Code" means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated documentation, interface definition files, scripts used to control compilation and installation of an Executable, or source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor's choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.12. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. Source Code License.

2.1 The Initial Developer Grant.

The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Code, to make, have made, use, practice, sell, and offer for

sale, and/or otherwise dispose of the Original Code (or portions thereof).

(c) the licenses granted in this Section 2.1(a) and (b) are effective on the date Initial Developer first distributes Original Code under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: 1) for code that You delete from the Original Code; 2) separate from the Original Code; or 3) for infringements caused by: i) the modification of the Original Code or ii) the combination of the Original Code with other software or devices.

2.2. Contributor Grant.

Subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor, to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: 1) Modifications made by that Contributor (or portions thereof); and 2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) the licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first makes Commercial Use of the Covered Code.

(d) notwithstanding Section 2.2(b) above, no patent license is granted: 1) for any code that Contributor has deleted from the Contributor Version; 2) separate from the Contributor Version; 3) for infringements caused by: i) third party modifications of Contributor Version or ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or 4) under Patent Claims infringed by Covered Code in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Application of License.

The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients' rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

3.2. Availability of Source Code.

Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.

3.3. Description of Modifications.

You must cause all Covered Code to which You contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

3.4. Intellectual Property Matters.

(a) Third Party Claims.

If Contributor has knowledge that a license under a third party's intellectual property rights is required to exercise the rights granted by such Contributor under Sections 2.1 or 2.2, Contributor must include a text file with the Source Code distribution titled

"LEGAL" which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If Contributor obtains such knowledge after the Modification is made available as described in Section 3.2, Contributor shall promptly modify the LEGAL file in all copies Contributor makes available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs.

If Contributor's Modifications include an application programming interface ("API") and Contributor has knowledge of patent licenses which are reasonably necessary to implement that API, Contributor must also include this information in the LEGAL file.

(c) Representations.

Contributor represents that, except as disclosed pursuant to Section 3.4(a) above, Contributor believes that Contributor's Modifications are Contributor's original creation(s) and/or Contributor has sufficient rights to grant the rights conveyed by this License.

3.5. Required Notices.

You must duplicate the notice in Exhibit A in each file of the Source Code. If it is not possible to put such notice in a particular Source Code file due to its structure, then You must include such notice in a location (such as a relevant directory) where a user would be likely to look for such a notice. If You created one or more Modification(s) You may add your name as a Contributor to the notice described in Exhibit A. You must also duplicate this License in any documentation for the Source Code where You describe recipients' rights or ownership rights relating to Covered Code. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. Distribution of Executable Versions.

You may distribute Covered Code in Executable form only if the

requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code or ownership rights under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.7. Larger Works.

You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Application of this License.

This License applies to code to which the Initial Developer has attached the notice in Exhibit A and to related Covered Code.

6. Versions of the License.

6.1. New Versions.

Sun Microsystems, Inc. ("Sun") may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.

6.2. Effect of New Versions.

Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Sun. No one other than Sun has the right to modify the terms applicable to Covered Code created under this License.

6.3. Derivative Works.

If You create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this License), You must: (a) rename Your license so that the phrases "Sun," "Sun Public License," or "SPL" or any confusingly similar phrase do not appear in your license (except to note that your license differs from this License) and (b) otherwise make it clear that Your version of the license contains terms which differ from the Sun Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. TERMINATION.

8.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall

survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

8.2. If You initiate litigation by asserting a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You file such action is referred to as "Participant") alleging that:

(a) such Participant's Contributor Version directly or indirectly infringes any patent, then any and all rights granted by such Participant to You under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively, unless if within 60 days after receipt of notice You either: (i) agree in writing to pay Participant a mutually agreeable reasonable royalty for Your past and future use of Modifications made by such Participant, or (ii) withdraw Your litigation claim with respect to the Contributor Version against such Participant. If within 60 days of notice, a reasonable royalty and payment arrangement are not mutually agreed upon in writing by the parties or the litigation claim is not withdrawn, the rights granted by Participant to You under Sections 2.1 and/or 2.2 automatically terminate at the expiration of the 60 day notice period specified above.

(b) any software, hardware, or device, other than such Participant's Contributor Version, directly or indirectly infringes any patent, then any rights granted to You by such Participant under Sections 2.1(b) and 2.2(b) are revoked effective as of the date You first made, used, sold, distributed, or had made, Modifications made by that Participant.

8.3. If You assert a patent infringement claim against Participant alleging that such Participant's Contributor Version directly or indirectly infringes any patent where such claim is resolved (such as by license or settlement) prior to the initiation of patent infringement litigation, then the reasonable value of the licenses granted by such Participant under Sections 2.1 or 2.2 shall be taken into account in determining the amount or value of any payment or license.

8.4. In the event of termination under Sections 8.1 or 8.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or any distributor hereunder prior to termination shall survive termination.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT

(INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein.

11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in the United States of America, any litigation relating to this License shall be subject to the jurisdiction of the Federal Courts of the Northern District of California, with venue lying in Santa Clara County, California, with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is

responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

13. MULTIPLE-LICENSED CODE.

Initial Developer may designate portions of the Covered Code as ?Multiple-Licensed?. ?Multiple-Licensed? means that the Initial Developer permits you to utilize portions of the Covered Code under Your choice of the alternative licenses, if any, specified by the Initial Developer in the file described in Exhibit A.

Exhibit A -Sun Public License Notice.

The contents of this file are subject to the Sun Public License Version 1.0 (the "License"); you may not use this file except in compliance with the License. A copy of the License is available at http://www.sun.com/

The Original Code is _____. The Initial Developer of the Original Code is _____. Portions created by _____ are Copyright (C)_____. All Rights Reserved.

Contributor(s): _____.

Alternatively, the contents of this file may be used under the terms of the _____ license (the ?[___] License?), in which case the provisions of [_____] License are applicable instead of those above. If you wish to allow use of your version of this file only under the terms of the [____] License and not to allow others to use your version of this file under the SPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the [___] License. If you do not delete the provisions above, a recipient may use your version of this file under either the SPL or the [___] License."

[NOTE: The text of this Exhibit A may differ slightly from the text of the notices in the Source Code files of the Original Code. You should use the text of this Exhibit A rather than the text found in the Original Code Source Code for Your Modifications.]

xml-commons/java/external/LICENSE.dom-documentation.txt $Id: LICENSE.dom-documentation.txt,v 1.2 2005/06/03 22:49:13 mrglavas Exp $

This license came from: http://www.w3.org/Consortium/Legal/copyright-documents-20021231

W3C DOCUMENT LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

Public documents on the W3C site are provided by the copyright holders under the following license. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231"
3. If it exists, the STATUS of the W3C document.

When space permits, inclusion of the full text of this NOTICE should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising

or publicity pertaining to this document or its contents without specific,
written prior permission. Title to copyright in this document will at all
times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002.
This version removes the copyright ownership notice such that this license can
be used with materials other than those owned by the W3C, moves information on
style sheets, DTDs, and schemas to the Copyright FAQ, reflects that ERCIM is
now a host of the W3C, includes references to this specific dated version of
the license, and removes the ambiguous grant of "use". See the older
formulation for the policy prior to this date. Please see our Copyright FAQ for
common questions about using materials from our site, such as the translating
or annotating specifications. Other questions about this notice can be directed
to site-policy@w3.org.

Joseph Reagle <site-policy@w3.org>

Last revised by Reagle $Date: 2005/06/03 22:49:13 $
xml-commons/java/external/LICENSE.sax.txt $Id: LICENSE.sax.txt,v 1.1 2002/01/31 23:26:48 curcuru Exp $

This license came from: http://www.megginson.com/SAX/copying.html
However please note future versions of SAX may be covered
under http://saxproject.org/?selected=pd

This page is now out of date -- see the new SAX site at
http://www.saxproject.org/ for more up-to-date
releases and other information. Please change your bookmarks.

SAX2 is Free!

I hereby abandon any property rights to SAX 2.0 (the Simple API for
XML), and release all of the SAX 2.0 source code, compiled code, and
documentation contained in this distribution into the Public Domain.
SAX comes with NO WARRANTY or guarantee of fitness for any
purpose.

David Megginson, david@megginson.com
2000-05-05

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of

the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works

that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A

PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

COMMON DEVELOPMENT AND DISTRIBUTION LICENSE Version 1.0 (CDDL-1.0)
(text)

1. Definitions.

1.1. Contributor means each individual or entity that creates or contributes to the creation of Modifications.

1.2. Contributor Version means the combination of the Original Software, prior Modifications used by a Contributor (if any), and the Modifications made by that particular Contributor.

1.3. Covered Software means (a) the Original Software, or (b) Modifications, or (c) the combination of files containing Original Software with files containing Modifications, in each case including portions thereof.

1.4. Executable means the Covered Software in any form other than Source Code.

1.5. Initial Developer means the individual or entity that first makes Original Software available under this License.

1.6. Larger Work means a work which combines Covered Software or portions thereof with code not governed by the terms of this License.

1.7. License means this document.

1.8. Licensable means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. Modifications means the Source Code and Executable form of any of the following:

A. Any file that results from an addition to, deletion from or modification of the contents of a file containing Original Software or previous Modifications;

B. Any new file that contains any part of the Original Software or previous Modification; or

C. Any new file that is contributed or otherwise made available under the terms of this License.

1.10. Original Software means the Source Code and Executable form of computer software code that is originally released under this License.

1.11. Patent Claims means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

1.12. Source Code means (a) the common form of computer software code in which modifications are made and

(b) associated documentation included in or with such code.

1.13. You (or Your) means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License. For legal entities, You includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, control means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants.

2.1. The Initial Developer Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, the Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Initial Developer, to use, reproduce, modify, display, perform, sublicense and distribute the Original Software (or portions thereof), with or without Modifications, and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using or selling of Original Software, to make, have made, use, practice, sell, and offer for sale, and/or otherwise dispose of the Original Software (or portions thereof).

(c) The licenses granted in Sections 2.1(a) and (b) are effective on the date Initial Developer first distributes or otherwise makes the Original Software available to a third party under the terms of this License.

(d) Notwithstanding Section 2.1(b) above, no patent license is granted: (1) for code that You delete from the Original Software, or (2) for infringements caused by: (i) the modification of the Original Software, or (ii) the combination of the Original Software with other software or devices.

2.2. Contributor Grant.

Conditioned upon Your compliance with Section 3.1 below and subject to third party intellectual property claims, each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by Contributor to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof), either on an unmodified basis, with other Modifications, as Covered Software and/or as part of a Larger Work; and

(b) under Patent Claims infringed by the making, using, or selling of Modifications made by that Contributor either alone and/or in combination with its Contributor Version (or portions of such combination), to make, use, sell, offer for sale, have made, and/or otherwise dispose of: (1) Modifications made by that Contributor (or portions thereof); and (2) the combination of Modifications made by that Contributor with its Contributor Version (or portions of such combination).

(c) The licenses granted in Sections 2.2(a) and 2.2(b) are effective on the date Contributor first distributes or otherwise makes the Modifications available to a third party.

(d) Notwithstanding Section 2.2(b) above, no patent license is granted: (1) for any code that Contributor has deleted from the Contributor Version; (2) for infringements caused by: (i) third party modifications of Contributor Version, or (ii) the combination of Modifications made by that Contributor with other software (except as part of the Contributor Version) or other devices; or (3) under Patent Claims infringed by Covered Software in the absence of Modifications made by that Contributor.

3. Distribution Obligations.

3.1. Availability of Source Code.

Any Covered Software that You distribute or otherwise make available in Executable form must also be made available in Source Code form and that Source Code form must be distributed only under the terms of this License. You must include a copy of this License with every copy of the Source Code form of the Covered Software You distribute or otherwise make available. You must inform recipients of any such Covered Software in Executable form as to how they can obtain such Covered Software in Source Code form in a reasonable manner on or through a medium customarily used for software exchange.

3.2. Modifications.

The Modifications that You create or to which You contribute are governed by the terms of this License. You represent that You believe Your Modifications are Your original creation(s) and/or You have sufficient rights to grant the rights conveyed by this License.

3.3. Required Notices.

You must include a notice in each of Your Modifications that identifies You as the Contributor of the Modification. You may not remove or alter any copyright, patent or trademark notices contained within the Covered Software, or any notices of licensing or any descriptive text giving attribution to any Contributor or the Initial Developer.

3.4. Application of Additional Terms.

You may not offer or impose any terms on any Covered Software in Source Code form that alters or restricts the applicable version of this License or the recipients rights hereunder. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, you may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear that any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.5. Distribution of Executable Versions.

You may distribute the Executable form of the Covered Software under the terms of this License or under the terms of a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable form does not attempt to limit or alter the recipients rights in the Source Code form from the rights set forth in this License. If You distribute the Covered Software in Executable form under a different license, You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or Contributor. You hereby

agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.6. Larger Works.

You may create a Larger Work by combining Covered Software with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Software.

4. Versions of the License.

4.1. New Versions.

Sun Microsystems, Inc. is the initial license steward and may publish revised and/or new versions of this License from time to time. Each version will be given a distinguishing version number. Except as provided in Section 4.3, no one other than the license steward has the right to modify this License.

4.2. Effect of New Versions.

You may always continue to use, distribute or otherwise make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. If the Initial Developer includes a notice in the Original Software prohibiting it from being distributed or otherwise made available under any subsequent version of the License, You must distribute and make the Covered Software available under the terms of the version of the License under which You originally received the Covered Software. Otherwise, You may also choose to use, distribute or otherwise make the Covered Software available under the terms of any subsequent version of the License published by the license steward.

4.3. Modified Versions.

When You are an Initial Developer and You want to create a new license for Your Original Software, You may create and use a modified version of this License if You: (a) rename the license and remove any references to the name of the license steward (except to note that the license differs from this License); and (b) otherwise make it clear that the license contains terms which differ from this License.

5. DISCLAIMER OF WARRANTY.

COVERED SOFTWARE IS PROVIDED UNDER THIS LICENSE ON AN AS IS BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED SOFTWARE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED SOFTWARE IS WITH YOU. SHOULD ANY COVERED SOFTWARE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

6. TERMINATION.

6.1. This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

6.2. If You assert a patent infringement claim (excluding declaratory judgment actions) against Initial Developer or a Contributor (the Initial Developer or Contributor against whom You assert such claim is referred to as Participant) alleging that the Participant Software (meaning the Contributor Version where the Participant is a Contributor or the Original Software where the Participant is the Initial Developer) directly or indirectly infringes any patent, then any and all rights granted directly or indirectly to You by such Participant, the Initial Developer (if the Initial Developer is not the Participant) and all Contributors under Sections 2.1 and/or 2.2 of this License shall, upon 60 days notice from Participant terminate prospectively and automatically at the expiration of such 60 day notice period, unless if within such 60 day period You withdraw Your claim with respect to the Participant Software against such Participant either unilaterally or pursuant to a written agreement with Participant.

6.3. In the event of termination under Sections 6.1 or 6.2 above, all end user licenses that have been validly granted by You or any distributor hereunder prior to termination (excluding licenses granted to You by any distributor) shall survive termination.

7. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL YOU, THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED SOFTWARE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOST PROFITS, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTYS NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

8. U.S. GOVERNMENT END USERS.

The Covered Software is a commercial item, as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of commercial computer software (as that term is defined at 48 C.F.R. 252.227-7014(a)(1)) and commercial computer software documentation as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Software with only those rights set forth herein. This U.S. Government Rights clause is in lieu of, and supersedes, any other FAR, DFAR, or other clause or provision that addresses Government rights in computer software under this License.

9. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable.

This License shall be governed by the law of the jurisdiction specified in a notice contained within the Original Software (except to the extent applicable law, if any, provides otherwise), excluding such jurisdictions conflict-of-law provisions. Any litigation relating to this License shall be subject to the jurisdiction of the courts located in the jurisdiction and venue specified in a notice contained within the Original Software, with the losing party responsible for costs, including, without limitation, court costs and reasonable attorneys fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License. You agree that You alone are responsible for compliance with the United States export administration regulations (and the export control laws and regulation of any other countries) when You use, distribute or otherwise make available any Covered Software.

10. RESPONSIBILITY FOR CLAIMS.

As between Initial Developer and the Contributors, each party is responsible for claims and damages arising, directly or indirectly, out of its utilization of rights under this License and You agree to work with Initial Developer and Contributors to distribute such responsibility on an equitable basis. Nothing herein is intended or shall be deemed to constitute any admission of liability.

1.133 xercesImpl 2.9.1

1.133.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed

as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this

License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all

other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and

subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed

as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the

Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

```
 ========================================================================
==  NOTICE file corresponding to section 4(d) of the Apache License,   ==
==  Version 2.0, in this case for the Apache Xerces Java distribution. ==
========================================================================
```

Apache Xerces Java

Copyright 1999-2007 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Portions of this software were originally based on the following:
- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
- voluntary contributions made by Paul Eng on behalf of the
Apache Software Foundation that were originally developed at iClick, Inc.,
software copyright (c) 1999.

Apache XML Commons Resolver
Copyright 2006 The Apache Software Foundation.

This product includes software developed at
The Apache Software Foundation http://www.apache.org/

Portions of this code are derived from classes placed in the
public domain by Arbortext on 10 Apr 2000. See:
http://www.arbortext.com/customer_support/updates_and_technical_notes/catalogs/docs/README.htm
==
== NOTICE file corresponding to section 4(d) of the Apache License, ==
== Version 2.0, in this case for the Apache Xalan Java distribution. ==
==

Apache Xalan (Xalan serializer)
Copyright 1999-2006 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Portions of this software was originally based on the following:
- software copyright (c) 1999-2002, Lotus Development Corporation.,
http://www.lotus.com.
- software copyright (c) 2001-2002, Sun Microsystems.,
http://www.sun.com.
- software copyright (c) 2003, IBM Corporation.,
http://www.ibm.com.

1.134 xml-apis-ext 1.3.04

1.134.1 Available under license :

Apache License

Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted"

means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and

attribution notices from the Source form of the Work,
excluding those notices that do not pertain to any part of
the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
distribution, then any Derivative Works that You distribute must
include a readable copy of the attribution notices contained
within such NOTICE file, excluding those notices that do not
pertain to any part of the Derivative Works, in at least one
of the following places: within a NOTICE text file distributed
as part of the Derivative Works; within the Source form or
documentation, if provided along with the Derivative Works; or,
within a display generated by the Derivative Works, if and
wherever such third-party notices normally appear. The contents
of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the

appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
===
== NOTICE file corresponding to section 4(d) of the Apache License, ==
== Version 2.0, in this case for the Apache xml-commons xml-apis ==
== distribution. ==
===

Apache XML Commons XML APIs
Copyright 1999-2009 The Apache Software Foundation.

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Portions of this software were originally based on the following:
- software copyright (c) 1999, IBM Corporation., http://www.ibm.com.
- software copyright (c) 1999, Sun Microsystems., http://www.sun.com.
- software copyright (c) 2000 World Wide Web Consortium, http://www.w3.org
xml-commons/java/external/LICENSE.dom-documentation.txt $Id: LICENSE.dom-documentation.txt 226215 2005-06-03 22:49:13Z mrglavas $

This license came from: http://www.w3.org/Consortium/Legal/copyright-documents-20021231

W3C DOCUMENT LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

Public documents on the W3C site are provided by the copyright holders under the following license. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.

http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231"
3. If it exists, the STATUS of the W3C document.

When space permits, inclusion of the full text of this NOTICE should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002. This version removes the copyright ownership notice such that this license can be used with materials other than those owned by the W3C, moves information on style sheets, DTDs, and schemas to the Copyright FAQ, reflects that ERCIM is now a host of the W3C, includes references to this specific dated version of the license, and removes the ambiguous grant of "use". See the older formulation for the policy prior to this date. Please see our Copyright FAQ for common questions about using materials from our site, such as the translating or annotating specifications. Other questions about this notice can be directed to site-policy@w3.org.

Joseph Reagle <site-policy@w3.org>

Last revised by Reagle $Date: 2005-06-03 18:49:13 -0400 (Fri, 03 Jun 2005) $ xml-commons/java/external/LICENSE.dom-software.txt $Id: LICENSE.dom-software.txt 734314 2009-01-14 03:33:27Z mrglavas $

This license came from: http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/java-binding.zip (COPYRIGHT.html)

W3C SOFTWARE NOTICE AND LICENSE
Copyright 2004 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.

The DOM bindings are published under the W3C Software Copyright Notice and License. The software license requires "Notice of any changes or modifications to the W3C files, including the date changes were made." Consequently, modified versions of the DOM bindings must document that they do not conform to the W3C standard; in the case of the IDL definitions, the pragma prefix can no longer be 'w3c.org'; in the case of the Java language binding, the package names can no longer be in the 'org.w3c' package.

Note: The original version of the W3C Software Copyright Notice and License could be found at http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.
3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR

CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or
publicity pertaining to the software without specific, written prior permission.
Title to copyright in this software and any associated documentation will at
all times remain with copyright holders.
xml-commons/java/external/LICENSE.sax.txt $Id: LICENSE.sax.txt 225954 2002-01-31 23:26:48Z curcuru $

This license came from: http://www.megginson.com/SAX/copying.html
However please note future versions of SAX may be covered
under http://saxproject.org/?selected=pd

This page is now out of date -- see the new SAX site at
http://www.saxproject.org/ for more up-to-date
releases and other information. Please change your bookmarks.

SAX2 is Free!

I hereby abandon any property rights to SAX 2.0 (the Simple API for
XML), and release all of the SAX 2.0 source code, compiled code, and
documentation contained in this distribution into the Public Domain.
SAX comes with NO WARRANTY or guarantee of fitness for any
purpose.

David Megginson, david@megginson.com
2000-05-05

1.135 xmlschema-core 2.0.1

1.135.1 Available under license :

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by

the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity
exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications,
including but not limited to software source code, documentation
source, and configuration files.

"Object" form shall mean any form resulting from mechanical
transformation or translation of a Source form, including but
not limited to compiled object code, generated documentation,
and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or
Object form, made available under the License, as indicated by a
copyright notice that is included in or attached to the work
(an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object
form, that is based on (or derived from) the Work and for which the
editorial revisions, annotations, elaborations, or other modifications
represent, as a whole, an original work of authorship. For the purposes
of this License, Derivative Works shall not include works that remain
separable from, or merely link (or bind by name) to the interfaces of,
the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained

within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be

liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apache WebServices - XmlSchema
Copyright 2004-2011 The Apache Software Foundation

This product includes software developed at
The Apache Software Foundation (http://www.apache.org/).

Portions Copyright 2006 International Business Machines Corp.

Portions Copyright (C) World Wide Web Consortium 2006, 2007 and licensed under the three-part BSD license.