



## **Network Cisco Network Connectivity Center Perl Reference Guide**

Cisco Network Connectivity Center

### **Corporate Headquarters**

Cisco Systems, Inc.  
170 West Tasman Drive  
San Jose, CA 95134-1706  
USA  
<http://www.cisco.com>  
Tel: 408 526-4000  
800 553-NETS (6387)  
Fax: 408 526-4100

Text Part Number: OL-6292-01



THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

CCSP, the Cisco Square Bridge logo, Cisco Unity, Follow Me Browsing, FormShare, and StackWise are trademarks of Cisco Systems, Inc.; Changing the Way We Work, Live, Play, and Learn, and iQuick Study are service marks of Cisco Systems, Inc.; and Aironet, ASIST, BPX, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Empowering the Internet Generation, Enterprise/Solver, EtherChannel, EtherFast, EtherSwitch, Fast Step, GigaDrive, GigaStack, HomeLink, Internet Quotient, IOS, IP/TV, iQ Expertise, the iQ logo, iQ Net Readiness Scorecard, LightStream, Linksys, MeetingPlace, MGX, the Networkers logo, Networking Academy, Network Registrar, *Packet*, PIX, Post-Routing, Pre-Routing, ProConnect, RateMUX, Registrar, ScriptShare, SlideCast, SMARTnet, StrataView Plus, SwitchProbe, TeleRouter, The Fastest Way to Increase Your Internet Quotient, TransPath, and VCO are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or Website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0406R)

*Network Cisco Network Connectivity Center Perl Reference Guide*  
Copyright © 2004 Cisco Systems, Inc. All rights reserved.

Copyright ©1996-2004 by System Management ARTS Incorporated. All rights reserved.

The Software and all intellectual property rights related thereto constitute trade secrets and proprietary data of SMARTS and any third party from whom SMARTS has received marketing rights, and nothing herein shall be construed to convey any title or ownership rights to you. Your right to copy the software and this documentation is limited by law. Making unauthorized copies, adaptations, or compilation works is prohibited and constitutes a punishable violation of the law. Use of the software is governed by its accompanying license agreement. The documentation is provided "as is" without warranty of any kind. In no event shall System Management ARTS Incorporated ("SMARTS") be liable for any loss of profits, loss of business, loss of use of data, interruption of business, or for indirect, special, incidental, or consequential damages of any kind, arising from any error in this documentation.

The InCharge products mentioned in this document are covered by one or more of the following U.S. patents or pending patent applications: 5,528,516, 5,661,668, 6,249,755, 10,124,881 and 60,284,860.

"InCharge," the InCharge logo, "SMARTS," the SMARTS logo, "Graphical Visualization," "Authentic Problem," "Codebook Correlation Technology," "Instant Results Technology," "InCharge Viewlet," and "Dashboard Viewlet" are trademarks or registered trademarks of System Management ARTS Incorporated. All other brand or product names are trademarks or registered trademarks of their respective companies or organizations..

---

Third-Party Software. The Software may include software of third parties from whom SMARTS has received marketing rights and is subject to some or all of the following additional terms and conditions:

#### Bundled Software

Sun Microsystems, Inc., Java(TM) Interface Classes, Java API for XML Parsing, Version 1.1. "Java" and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. SMARTS is independent of Sun Microsystems, Inc.

#### W3C IPR Software

Copyright © 2001-2003 World Wide Web Consortium (<http://www.w3.org>), (Massachusetts Institute of Technology (<http://www.lcs.mit.edu>), Institut National de Recherche en Informatique et en Automatique (<http://www.inria.fr>), Keio University (<http://www.keio.ac.jp>)). All rights reserved (<http://www.w3.org/Consortium/Legal/>). Note: The original version of the W3C Software Copyright Notice and License can be found at <http://www.w3.org/Consortium/Legal/copyright-software-19980720>.

#### The Apache Software License, Version 1.1

Copyright ©1999-2003 The Apache Software Foundation. All rights reserved. Redistribution and use of Apache source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of Apache source code must retain the above copyright notice, this list of conditions and the Apache disclaimer as written below.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the Apache disclaimer as written below 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 "The Jakarta Project", "Tomcat", "Xalan", "Xerces", and "Apache Software Foundation" must not be used to endorse or promote products derived from Apache software without prior written permission. For written permission, please contact [apache@apache.org](mailto:apache@apache.org).
5. Products derived from this Apache software may not be called "Apache," nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

**APACHE DISCLAIMER:** THIS APACHE SOFTWARE FOUNDATION 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 Apache software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright © 1999, Lotus Development Corporation., <http://www.lotus.com>. For information on the Apache Software Foundation, please see <http://www.apache.org>.

#### FLEXlm Software

© 1994 - 2003, Macrovision Corporation. All rights reserved. "FLEXlm" is a registered trademark of Macrovision Corporation. For product and legal information, see <http://www.macrovision.com/solutions/esd/flexlm/flexlm.shtml>.

#### JfreeChart – Java library for GIF generation

The Software is a "work that uses the library" as defined in GNU Lesser General Public License Version 2.1, February 1999 Copyright © 1991, 1999 Free Software Foundation, Inc., and is provided "AS IS" WITHOUT WARRANTY OF ANY KIND 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. 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 IN THE ABOVE-REFERENCED LICENSE 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. JfreeChart library (included herein as .jar files) is provided in accordance with, and its use is covered by the GNU Lesser General Public License Version 2.1, which is set forth at <http://www.object-refinery.com/lgpl.html>.

#### BMC – product library

The Software contains technology (product library or libraries) owned by BMC Software, Inc. ("BMC Technology"). BMC Software, Inc., its affiliates and licensors (including SMARTS) hereby disclaim all representations, warranties and liability for the BMC Technology.

#### Crystal Decisions Products

The Software may contain certain software and related user documentation (e.g., Crystal Enterprise Professional, Crystal Reports Professional and/or Crystal Analysis Professional) that are owned by Crystal Decisions, Inc., 895 Emerson Street, Palo Alto, CA 94301 ("Crystal Decisions"). All such software products are the technology of Crystal Decisions. The use of all Crystal Decisions software products is subject to a separate license agreement included with the Software electronically, in written materials, or both. YOU MAY NOT USE THE CRYSTAL DECISIONS SOFTWARE UNLESS AND UNTIL YOU READ, ACKNOWLEDGE AND ACCEPT THE TERMS AND CONDITIONS OF THE CRYSTAL DECISIONS' SOFTWARE LICENSE AGREEMENT. IF YOU DO NOT ACCEPT THE TERMS AND CONDITIONS OF THE CRYSTAL DECISIONS' SOFTWARE LICENSE, YOU MAY RETURN, WITHIN THIRTY (30) DAYS OF PURCHASE, THE MEDIA PACKAGE AND ALL ACCOMPANYING ITEMS (INCLUDING WRITTEN MATERIALS AND BINDERS OR OTHER CONTAINERS) RELATED TO THE CRYSTAL DECISIONS' TECHNOLOGY, TO SMARTS FOR A FULL REFUND; OR YOU MAY WRITE, CRYSTAL WARRANTIES, P.O. BOX 67427, SCOTTS VALLEY, CA 95067, U.S.A.

#### GNU eTeks PJA Toolkit

Copyright © 2000-2001 Emmanuel PUYBARET/eTeks [info@eteks.com](mailto:info@eteks.com). All Rights Reserved.

The eTeks PJA Toolkit is resident on the CD on which the Software was delivered to you. Additional information is available at eTeks' web site: <http://www.eteks.com>. The eTeks PJA Toolkit program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation; version 2 of the License. The full text of the applicable GNU GPL is available for viewing at <http://www.gnu.org/copyleft/gpl.txt>. You may also request a copy of the GPL from the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. The eTeks PJA Toolkit 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.

For a period of three years from the date of your license for the Software, you are entitled to receive under the terms of Sections 1 and 2 of the GPL, for a charge no more than SMARTS' cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code for the GNU eTeks PJA Toolkit provided to you hereunder by requesting such code from SMARTS in writing: Attn: Customer Support, SMARTS, 44 South Broadway, White Plains, New York 10601.

#### IBM Runtime for AIX

The Software contains the IBM Runtime Environment for AIX(R), Java™ 2 Technology Edition Runtime Modules © Copyright IBM Corporation 1999, 2000 All Rights Reserved.

#### HP-UX Runtime Environment for the Java™ 2 Platform

The Software contains the HP-UX Runtime for the Java™ 2 Platform, distributed pursuant to and governed by Hewlett-Packard Co. ("HP") software license terms set forth in detail at: <http://www.hp.com>. Please check the Software to determine the version of Java runtime distributed to you.

#### DataDirect Technologies

Portions of this software are copyrighted by DataDirect Technologies, 1991-2002.

#### NetBSD

Copyright © 2001 Christopher G. Demetriou. 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. All advertising materials mentioning features or use of this software must display the following acknowledgement:  
This product includes software developed for the NetBSD Project. See <http://www.netbsd.org/> for information about NetBSD.
4. 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. <<Id: LICENSE, v 1.2 2000/06/14 15:57:33 cgd Exp>>

#### RSA Data Security, Inc.

Copyright © 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved. License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function. License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work. RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind. These notices must be retained in any copies of any part of this documentation and/or software.

#### AES

Copyright © 2003, Dr Brian Gladman <brg@gladman.me.uk>, Worcester, UK. All rights reserved.

#### License Terms:

The free distribution and use of this software in both source and binary form is allowed (with or without changes) provided that:

1. distributions of this source code include the above copyright notice, this list of conditions and the following disclaimer;
2. distributions in binary form include the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other associated materials;
3. the copyright holder's name is not used to endorse products built using this software without specific written permission.

ALTERNATIVELY, provided that this notice is retained in full, this product may be distributed under the terms of the GNU General Public License (GPL), in which case the provisions of the GPL apply INSTEAD OF those given above.

Disclaimer: This software is provided 'as is' with no explicit or implied warranties in respect of its properties, including, but not limited to, correctness and/or fitness for purpose. Issue Date: 26/08/2003

# Contents

Preface	xiii
Intended Audience	xiii
Prerequisites	xiii
Document Organization	xiv
Documentation Conventions	xv
Additional Resources	xv
Commands	xv
Documentation	xvi
Common Abbreviations and Acronyms	xvi
Obtaining Documentation	xvii
Cisco.com	xvii
Ordering Documentation	xvii
Documentation Feedback	xviii
Obtaining Technical Assistance	xviii
Cisco Technical Support Website	xviii
Submitting a Service Request	xix
Definitions of Service Request Severity	xix
Obtaining Additional Publications and Information	xx
1 Introduction	1
Fundamental Concepts	1
<b>Classes</b>	<b>2</b>
<b>Instances</b>	<b>2</b>
<b>Properties - Attributes and Relationships</b>	<b>2</b>
<b>Operations</b>	<b>2</b>
<b>Events</b>	<b>3</b>
<b>Primitives - Basic CNCC Manager Interface</b>	<b>3</b>
Overview of a Simple API Perl Script	4
Using Primitives and Object References	5

Event Subscription	6
Registering an Observer	7
Subscribing to Notifications.	8
Receiving Notifications	8
2 Object	17
Name	17
Synopsis	17
Description	17
Functions and Methods	20
<b>object</b>	<b>20</b>
<b>get</b>	<b>20</b>
<b>get_t</b>	<b>21</b>
<b>put</b>	<b>22</b>
<b>isNull</b>	<b>22</b>
<b>invoke</b>	<b>22</b>
<b>invoke_t</b>	<b>23</b>
<b>insertElement</b>	<b>23</b>
<b>removeElement</b>	<b>23</b>
<b>delete</b>	<b>24</b>
<b>notify</b>	<b>24</b>
<b>clear</b>	<b>24</b>
<b>countElements</b>	<b>24</b>
3 Session	25
Name	25
Synopsis	25
Description	26
Function Groups	26
<b>Session Management</b>	<b>26</b>
<b>InCharge Primitives</b>	<b>26</b>
<b>Utility Functions</b>	<b>26</b>
<b>Compatibility Functions</b>	<b>27</b>
Error Handling	27

---

Session Management Functions	27
<b>new</b>	27
<b>init</b>	29
<b>broken</b>	30
<b>reattach</b>	30
<b>detach</b>	30
<b>observer</b>	31
<b>receiveEvent</b>	31
<b>object</b>	32
<b>create</b>	33
<b>callPrimitive</b>	33
Utility Functions	34
<b>TYPE Function</b>	34
<b>getFileno</b>	34
<b>getProtocolVersion</b>	34
<b>primitiveIsAvailable</b>	35
Compatibility Functions	35
<b>save</b>	35
<b>put</b>	35
<b>invoke</b>	36
<b>invoke_t</b>	37
<b>findInstances</b>	38
<b>getCauses</b>	38
<b>getClosure</b>	39
<b>getExplains</b>	40
<b>getExplainedBy</b>	40
<b>subscribe and unsubscribe</b>	41
<b>transaction, abortTxn and commitTxn</b>	42
<b>delete</b>	44
<b>getEventType</b>	44
<b>getServerName</b>	45
<b>insertElement</b>	45
<b>removeElement</b>	45

4	Primitives	47
	Name	47
	Description	47
	Primitive Naming Conventions	47
	Primitive Calling Conventions	48
	Error Handling	49
	Error codes	51
	<b>Data Types</b>	<b>52</b>
	<b>\$session</b>	<b>52</b>
	<b>\$object</b>	<b>52</b>
	<b>@objects</b>	<b>53</b>
	<b>\$symptom, @symptoms</b>	<b>53</b>
	<b>\$symptomData, @symptomData</b>	<b>54</b>
	<b>\$type, @types</b>	<b>54</b>
	<b>\$freshness</b>	<b>56</b>
	Primitives	56
	<b>classExists</b>	<b>56</b>
	<b>consistencyUpdate</b>	<b>56</b>
	<b>correlate</b>	<b>56</b>
	<b>countChildren</b>	<b>57</b>
	<b>countClassInstances</b>	<b>57</b>
	<b>countClasses</b>	<b>57</b>
	<b>countElements</b>	<b>57</b>
	<b>countInstances</b>	<b>57</b>
	<b>countLeafInstances</b>	<b>57</b>
	<b>countf</b>	<b>58</b>
	<b>createInstance</b>	<b>58</b>
	<b>deleteInstance</b>	<b>58</b>
	<b>deleteObserver</b>	<b>58</b>
	<b>eventIsExported</b>	<b>59</b>
	<b>execute</b>	<b>59</b>
	<b>executeProgram</b>	<b>59</b>
	<b>exists</b>	<b>59</b>
	<b>findInstances_P</b>	<b>59</b>
	<b>forceNotify</b>	<b>60</b>



---

<b>get</b>	<b>60</b>
<b>get_t and get_T</b>	<b>61</b>
<b>getAggregationEvents</b>	<b>61</b>
<b>getAllEventNames</b>	<b>61</b>
<b>getAllInstances</b>	<b>61</b>
<b>getAllProperties and getAllProperties_t</b>	<b>62</b>
<b>getArgDirection</b>	<b>62</b>
<b>getArgType</b>	<b>63</b>
<b>getAttributes</b>	<b>63</b>
<b>getAttributeNames</b>	<b>63</b>
<b>getAttributeTypes</b>	<b>63</b>
<b>getByKey</b>	<b>64</b>
<b>getByKey_t and getByKey_T</b>	<b>64</b>
<b>getByKeyf</b>	<b>64</b>
<b>getByKeyf_t and getByKeyf_T</b>	<b>64</b>
<b>getChildren</b>	<b>65</b>
<b>getClassDescription</b>	<b>65</b>
<b>getClassHierarchy</b>	<b>65</b>
<b>getClassInstances</b>	<b>65</b>
<b>getClasses</b>	<b>66</b>
<b>getCorrelationParameters</b>	<b>66</b>
<b>getEnumVals</b>	<b>67</b>
<b>getEvents</b>	<b>67</b>
<b>getEventCauses</b>	<b>67</b>
<b>getEventClassName</b>	<b>68</b>
<b>getEventDescription</b>	<b>68</b>
<b>getEventExplainedBy</b>	<b>68</b>
<b>getEventExported</b>	<b>68</b>
<b>getEventNames</b>	<b>69</b>
<b>getEventSymptoms</b>	<b>69</b>
<b>getEventType_P</b>	<b>69</b>
<b>getInstances</b>	<b>70</b>
<b>getInstrumentationType</b>	<b>70</b>
<b>getLeafInstances</b>	<b>70</b>

<b>getLibraries</b>	<b>70</b>
<b>getModels</b>	<b>71</b>
<b>getMultipleProperties and getMultipleProperties_t</b>	<b>71</b>
<b>getObserverId</b>	<b>71</b>
<b>getOpArgType</b>	<b>71</b>
<b>getOpArgs</b>	<b>71</b>
<b>getOpDescription</b>	<b>72</b>
<b>getOperationArguments</b>	<b>72</b>
<b>getOperationArgumentType</b>	<b>72</b>
<b>getOperationDescription</b>	<b>72</b>
<b>getOperationFlag</b>	<b>72</b>
<b>getOperationReturnType</b>	<b>72</b>
<b>getOperations</b>	<b>72</b>
<b>getOpFlag</b>	<b>73</b>
<b>getOpNames</b>	<b>73</b>
<b>getOpReturnType</b>	<b>73</b>
<b>getParentClass</b>	<b>74</b>
<b>getProblemClosure</b>	<b>74</b>
<b>getProblemExplanation</b>	<b>74</b>
<b>getProblemNames</b>	<b>74</b>
<b>getProblemSymptomState</b>	<b>74</b>
<b>getPrograms</b>	<b>75</b>
<b>getPropAccess</b>	<b>75</b>
<b>getPropDescription</b>	<b>75</b>
<b>getProperties</b>	<b>76</b>
<b>getPropertyDescription</b>	<b>76</b>
<b>getProperties</b>	<b>76</b>
<b>getPropertyType</b>	<b>76</b>
<b>getPropIsReadOnly</b>	<b>76</b>
<b>getPropIsRelationship</b>	<b>77</b>
<b>getPropIsRequired</b>	<b>77</b>
<b>getPropNames</b>	<b>77</b>
<b>getPropRange</b>	<b>77</b>
<b>getPropType</b>	<b>77</b>

---

<b>getPropertySubscriptionState</b>	<b>78</b>
<b>getRelatedClass</b>	<b>78</b>
<b>getRelationNames</b>	<b>78</b>
<b>getRelations</b>	<b>78</b>
<b>getRelationTypes</b>	<b>79</b>
<b>getReverseRelation</b>	<b>79</b>
<b>getSubscriptionState</b>	<b>79</b>
<b>getThreads</b>	<b>79</b>
<b>getf</b>	<b>80</b>
<b>getf_t and getf_T</b>	<b>80</b>
<b>getfAllProperties and getfAllProperties_t</b>	<b>81</b>
<b>getfMultipleProperties and getfMultipleProperties_t</b>	<b>81</b>
<b>hasRequiredProps</b>	<b>81</b>
<b>insertElement_P</b>	<b>81</b>
<b>instanceExists</b>	<b>82</b>
<b>invoke</b>	<b>82</b>
<b>invoke_t and invoke_T</b>	<b>82</b>
<b>invokeOperation</b>	<b>82</b>
<b>invokeOperation_t and invokeOperation_T</b>	<b>83</b>
<b>isAbstract</b>	<b>83</b>
<b>isBaseOf</b>	<b>84</b>
<b>isBaseOfOrProxy</b>	<b>84</b>
<b>isInstrumented</b>	<b>84</b>
<b>isMember</b>	<b>84</b>
<b>isMemberByKey</b>	<b>85</b>
<b>isMemberByKeyf</b>	<b>85</b>
<b>isMemberf</b>	<b>85</b>
<b>isSubscribed</b>	<b>85</b>
<b>loadLibrary</b>	<b>86</b>
<b>loadModel</b>	<b>86</b>
<b>loadProgram</b>	<b>86</b>
<b>noop</b>	<b>86</b>
<b>notify</b>	<b>86</b>
<b>ping</b>	<b>86</b>

<b>propertySubscribe</b>	<b>86</b>
<b>propertySubscribeAll</b>	<b>87</b>
<b>propertyUnsubscribe</b>	<b>87</b>
<b>propertyUnsubscribeAll</b>	<b>87</b>
<b>purgeObserver</b>	<b>87</b>
<b>put_P</b>	<b>87</b>
<b>quit</b>	<b>88</b>
<b>removeElement_P</b>	<b>88</b>
<b>removeElementByKey</b>	<b>89</b>
<b>restoreRepository</b>	<b>89</b>
<b>setCorrelationParameters</b>	<b>89</b>
<b>shutdown</b>	<b>89</b>
<b>storeAllRepository</b>	<b>89</b>
<b>storeClassRepository</b>	<b>90</b>
<b>subscribeEvent</b>	<b>90</b>
<b>subscribeAll</b>	<b>90</b>
<b>topologySubscribe</b>	<b>90</b>
<b>topologyUnsubscribe</b>	<b>90</b>
<b>transactionAbort</b>	<b>91</b>
<b>transactionCommit</b>	<b>91</b>
<b>transactionStart</b>	<b>91</b>
<b>unsubscribeAll</b>	<b>92</b>
<b>unsubscribeEvent</b>	<b>92</b>
 Index	 93

# Preface

## Intended Audience

This document is intended for software developers using the Perl programming language to implement InCharge subscription and adapter applications.

## Prerequisites

It is beneficial to have an understanding of the basic platform operation, functionality, and architecture, and the MODEL programming language, which are available through the *SMARTS Educational Services* tutorials.

Proficiency in the Perl programming language is required, along with familiarity with the language's development environment and tools.

Readers of this document should also be familiar with basic programming and communication concepts and software development methodology, especially as they relate to distributed systems.

# Document Organization

This guide consists of the following chapters.

1. INTRODUCTION	Gives an overview of the features accessible to a programmatic interface and a discussion of how the Perl programming language may be used to interact with a CNCC Manager.
2.OBJECT	Provides a description of the API facilities available that allow objects in the repository to be manipulated in an object-oriented style, similar to the ASL language.
3.SESSION	Provides a description of the API facilities available to create Perl scripts to establish client/server connections and to obtain object references, which can be used to manipulate the objects in the topology.
4.PRIMITIVES	Provides a description of the Perl low-level remote calls supported by the API. These primitives provide the standard protocol between client application

Table 1: **Document Organization**

# Documentation Conventions

Several conventions may be used in this document as shown in Table 2.

CONVENTION	EXPLANATION
<code>sample code</code>	Indicates code fragments and examples in Courier font
<b>keyword</b>	Indicates commands, keywords, literals, and operators in bold
<code>%</code>	Indicates C shell prompt
<code>#</code>	Indicates C shell superuser prompt
<code>&lt;parameter&gt;</code>	Indicates a user-supplied value or a list of non-terminal items in angle brackets
<code>[option]</code>	Indicates optional terms in brackets
<i>/InCharge</i>	Indicates directory path names in italics
<b><i>yourDomain</i></b>	Indicates a user-specific or user-supplied value in bold, italics
<i>File &gt; Open</i>	Indicates a menu path in italics

Table 2: **Documentation Conventions**

Directory path names are shown with forward slashes (/). Users of the Windows operating systems should substitute back slashes (\) for forward slashes.

Also, if there are figures illustrating consoles in this document, they represent the consoles as they appear in Windows. Under UNIX, the consoles appear with slight differences. For example, in views that display items in a tree hierarchy such as the Topology Browser, a plus sign displays for Windows and an open circle displays for UNIX.

Finally, unless otherwise specified, the term CNCC Manager is used to refer to NCM programs such as Domain Managers, Global Managers, and adapters.

## Additional Resources

In addition to this manual, Cisco provides the following resources.

### Commands

Descriptions of commands are available as HTML pages. The *index.html* file, which provides an index to the various commands, is located in the ***BASEDIR/smarts/doc/html/usage*** directory.

## Documentation

Readers of this manual may find other documentation (also available in the *BASEDIR/smarts/doc/pdf* directory) helpful.

### Network Connectivity Monitor Documentation

The following documents are product independent and thus relevant to users of Network Connectivity Monitor products:

- *Release Notes for Network Connectivity Monitor 1.1*
- *Network Connectivity Monitor Documentation Roadmap*
- *Network Connectivity Monitor System Administration Guide*
- *ICIM Reference*
- *InCharge ASL Reference Guide*
- *Cisco Network Connectivity Center Perl Reference Guide*

### Software Development Kit Documentation

The following documents are relevant to users of the Software Development Kit.

- *InCharge Software Development Kit Remote API for Java* (in HTML format)
- *InCharge Software Development Kit Remote API Programmer's Guide*
- *ICIM Reference* (in HTML format)
- *InCharge Software Development Kit MODEL Reference Guide*

## Common Abbreviations and Acronyms

The following lists common abbreviations and acronyms that are used in the InCharge guides.

ASL	Adapter Scripting Language
CDP	Cisco Discovery Protocol
ICIM	InCharge Common Information Model
ICMP	Internet Control Message Protocol
IP	Internet Protocol
MSFC	Multilayer Switch Feature Card
MIB	Management Information Base



MODEL	Managed Object Definition Language
RSFC	Router Switch Feature Card
RSM	Router Switch Module
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
VLAN	Virtual Local Area Network

## Obtaining Documentation

Cisco documentation and additional literature are available on Cisco.com. Cisco also provides several ways to obtain technical assistance and other technical resources. These sections explain how to obtain technical information from Cisco Systems.

### Cisco.com

You can access the most current Cisco documentation at this URL:

<http://www.cisco.com/univercd/home/home.htm>

You can access the Cisco website at this URL:

<http://www.cisco.com>

You can access international Cisco websites at this URL:

[http://www.cisco.com/public/countries\\_languages.shtml](http://www.cisco.com/public/countries_languages.shtml)

### Ordering Documentation

You can find instructions for ordering documentation at this URL:

[http://www.cisco.com/univercd/cc/td/doc/es\\_inpk/pdi.htm](http://www.cisco.com/univercd/cc/td/doc/es_inpk/pdi.htm)

You can order Cisco documentation in these ways:

- Registered Cisco.com users (Cisco direct customers) can order Cisco product documentation from the Ordering tool:

<http://www.cisco.com/en/US/partner/ordering/index.shtml>

- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco Systems Corporate Headquarters (California, USA) at 408 526-7208 or, elsewhere in North America, by calling 800 553-NETS (6387).

## Documentation Feedback

You can send comments about technical documentation to [bug-doc@cisco.com](mailto:bug-doc@cisco.com).

You can submit comments by using the response card (if present) behind the front cover of your document or by writing to the following address:

Cisco Systems  
Attn: Customer Document Ordering  
170 West Tasman Drive  
San Jose, CA 95134-9883

We appreciate your comments.

## Obtaining Technical Assistance

For all customers, partners, resellers, and distributors who hold valid Cisco service contracts, Cisco Technical Support provides 24-hour-a-day, award-winning technical assistance. The Cisco Technical Support Website on Cisco.com features extensive online support resources. In addition, Cisco Technical Assistance Center (TAC) engineers provide telephone support. If you do not hold a valid Cisco service contract, contact your reseller.

## Cisco Technical Support Website

The Cisco Technical Support Website provides online documents and tools for troubleshooting and resolving technical issues with Cisco products and technologies. The website is available 24 hours a day, 365 days a year at this URL:

<http://www.cisco.com/techsupport>

Access to all tools on the Cisco Technical Support Website requires a Cisco.com user ID and password. If you have a valid service contract but do not have a user ID or password, you can register at this URL:

<http://tools.cisco.com/RPF/register/register.do>

## Submitting a Service Request

Using the online TAC Service Request Tool is the fastest way to open S3 and S4 service requests. (S3 and S4 service requests are those in which your network is minimally impaired or for which you require product information.) After you describe your situation, the TAC Service Request Tool automatically provides recommended solutions. If your issue is not resolved using the recommended resources, your service request will be assigned to a Cisco TAC engineer. The TAC Service Request Tool is located at this URL:

<http://www.cisco.com/techsupport/servicerequest>

For S1 or S2 service requests or if you do not have Internet access, contact the Cisco TAC by telephone. (S1 or S2 service requests are those in which your production network is down or severely degraded.) Cisco TAC engineers are assigned immediately to S1 and S2 service requests to help keep your business operations running smoothly.

To open a service request by telephone, use one of the following numbers:

Asia-Pacific: +61 2 8446 7411 (Australia: 1 800 805 227)

EMEA: +32 2 704 55 55

USA: 1 800 553 2447

For a complete list of Cisco TAC contacts, go to this URL:

<http://www.cisco.com/techsupport/contacts>

## Definitions of Service Request Severity

To ensure that all service requests are reported in a standard format, Cisco has established severity definitions.

**Severity 1 (S1)**—Your network is “down,” or there is a critical impact to your business operations. You and Cisco will commit all necessary resources around the clock to resolve the situation.

**Severity 2 (S2)**—Operation of an existing network is severely degraded, or significant aspects of your business operation are negatively affected by inadequate performance of Cisco products. You and Cisco will commit full-time resources during normal business hours to resolve the situation.

Severity 3 (S3)—Operational performance of your network is impaired, but most business operations remain functional. You and Cisco will commit resources during normal business hours to restore service to satisfactory levels.

Severity 4 (S4)—You require information or assistance with Cisco product capabilities, installation, or configuration. There is little or no effect on your business operations.

## Obtaining Additional Publications and Information

Information about Cisco products, technologies, and network solutions is available from various online and printed sources.

- Cisco Marketplace provides a variety of Cisco books, reference guides, and logo merchandise. Visit Cisco Marketplace, the company store, at this URL:  
<http://www.cisco.com/go/marketplace/>
- The Cisco *Product Catalog* describes the networking products offered by Cisco Systems, as well as ordering and customer support services. Access the Cisco Product Catalog at this URL:  
<http://cisco.com/univercd/cc/td/doc/pcat/>
- *Cisco Press* publishes a wide range of general networking, training and certification titles. Both new and experienced users will benefit from these publications. For current Cisco Press titles and other information, go to Cisco Press at this URL:  
<http://www.ciscopress.com>
- *Packet* magazine is the Cisco Systems technical user magazine for maximizing Internet and networking investments. Each quarter, Packet delivers coverage of the latest industry trends, technology breakthroughs, and Cisco products and solutions, as well as network deployment and troubleshooting tips, configuration examples, customer case studies, certification and training information, and links to scores of in-depth online resources. You can access Packet magazine at this URL:  
<http://www.cisco.com/packet>

- *iQ Magazine* is the quarterly publication from Cisco Systems designed to help growing companies learn how they can use technology to increase revenue, streamline their business, and expand services. The publication identifies the challenges facing these companies and the technologies to help solve them, using real-world case studies and business strategies to help readers make sound technology investment decisions. You can access iQ Magazine at this URL:

<http://www.cisco.com/go/iqmagazine>

- *Internet Protocol Journal* is a quarterly journal published by Cisco Systems for engineering professionals involved in designing, developing, and operating public and private internets and intranets. You can access the Internet Protocol Journal at this URL:

<http://www.cisco.com/ipj>

- World-class networking training is available from Cisco. You can view current offerings at this URL:

<http://www.cisco.com/en/US/learning/index.html>



# Introduction

The Remote Application Programming Interface (API) for Perl allows developers to create Perl scripts that connect to CNCC Servers as CNCC Clients to exchange information, manipulate data, or drive CNCC Manager actions.

The API provides access to all CNCC Manager features using a syntax and logic that mirrors that available through the Adapter Scripting Language (ASL) and the dmctl utility in a way that is natural to Perl developers.

---

**Note:** This API is compatible with CNCC Manager Version 6.0 and later.

---

The API runs on Windows 2000 and Unix platforms that support Perl 5.6.1 and Perl 5.8.0 and the IO::Socket module.

## Fundamental Concepts

In order to create scripts that interact with a CNCC Manager, it is necessary to understand how the CNCC Manager is configured.

The script creates, deletes and interacts with instances of objects defined using the MODEL language. The MODEL language is an object-oriented language used to construct a data model to describe a managed domain. The language is used to define a set of classes and the attributes, relationships, operations, and events that are associated with the classes.

## Classes

Classes describe the objects that are modeled for use in a CNCC Manager. For example, Router is a class, and all routers that are managed by a CNCC domain are represented as Router objects in the domain. Every object in a class shares the same set of attributes, although the values of the attributes differ. Hence, every router has an IP address, an attribute, but the actual addresses are different. PowerSupply is also a class but power supplies do not have IP addresses. However, the event power outage is relevant to the PowerSupply class but not to routers. Therefore, a model Class is a grouping of all objects which are similar in nature but not in detail.

Every class has a number of properties, events and operations defined for them. The API provides functions for obtaining details of these definitions.

## Instances

Object instances are specific occurrences of a class. For example, a class might describe a human and an instance of the class could be an object named Bill.

## Properties - Attributes and Relationships

Every instance in a CNCC domain has a set of properties associated with it. These are values that describe the object. There are two distinct types of class property supported by CNCC.

### 1 Attributes

Attributes describe a class and for an instance of the class include information about its present state. Examples of attributes include an element's name and a counter that counts the number of packets traversing an interface. Attributes are simple strings, integers, booleans or enumerations.

### 2 Relationships

Relationships define how instances are related to other instances. Relationships can be one-to-one, one-to-many, many-to-one or many-to-many. When only a single instance can be related to another instance, or instances, it is known as a *relationship*. When multiple instances can be related to another instance, or instances, it is known as a *relationshipset*.

## Operations

Operations are actions that are specific to a class of object. For example, it makes sense to get the associated network adapter name for a MAC address but not for a Router.



The API provides a mechanism for invoking these class-specific actions, passing information to them via arguments, and obtaining the results of the action.

## Events

Events describe the failures that can occur for a class, the symptoms these failures cause, and the effect of failures. Symptoms can be local, observed in the instance of the class, or propagated, observed in instances related to the failing instances.

## Primitives - Basic CNCC Manager Interface

Primitives are Perl functions that provide the basic interface between a client application and the CNCC Manager.

A number of these are likely to be used directly by scripts and will be familiar to ASL developers. These include *getInstances()*, *getChildren()*, *getExplainedBy()*.

Others are normally hidden from view because higher level features can be used instead, which ultimately call the primitives. Example of primitives that are not normally used directly are *get()* and *invoke()*. These are the CNCC calls that allow an instance's properties to be queried and its operations to be called. In both the API for Perl and ASL, these calls are normally invoked by using a classic object-oriented syntax.

For more information about the data structures, refer to the *InCharge Software Development Kit ASL Reference Guide*, chapter 7.

The Perl API is implemented as a set of Perl modules, which individual Perl scripts may access via the familiar 'use' directive. *InCharge::session* and *InCharge::object* offer the principal interface, respectively providing connection sessions to CNCC Managers and access to objects within those managers. Simultaneous sessions to multiple CNCC Managers may be established within a single Perl script. Properties and methods of objects within those Managers may be accessed as with C++, offering somewhat broader functionality than that afforded by ASL.

# Overview of a Simple API Perl Script

The general approach to writing a script using the API for CNCC is to follow these basic steps.

## 1 Open a session

Initialize a session, and obtain a reference to it (using either *InCharge::session->init()* or *InCharge::session->new()*, as appropriate).

```
use InCharge::session;
$session = InCharge::session->init( );
```

You would choose the *init()* method if you want the script's user to supply the session connection details on the command line (such as the broker, domain name, user name and password). The *new()* method can be used if you want your script to have greater control over the details used to establish the connection.

## 2 Work with the domain

Call the primitives required using the session reference obtained in step 1, and manipulate the data. For example,

```
foreach $class ( sort $session->getClasses() ) {
    foreach $inst (
        sort $session->getInstances($class)
    ) {
        print $class . "::$" . $inst . "\n";
    }
}
```

## 3 Close the session

Once the script has finished working with the domain, the session should be closed.

```
$session->detach( );
```

Where access to the operations or properties of CNCC domain objects (such as Routers and Interfaces) is required, you use the features of the *InCharge::object* module. The script obtains an *InCharge::object* reference, and then uses it to access the required information. For example.

## 1 Establish a session

```
use InCharge::session;
$session = InCharge::session->init();
```

## 2 Obtain an object reference

Before an object in the domain can be accessed, the script needs to obtain an `InCharge::object` reference to the object of interest using the `object()` method of the session handle.

```
$obj = $session->object( "Router::gw1" );
```

### 3 Manipulate the object

The reference obtained in step 2 can now be used to access the properties and operations of the object. Properties can be accessed using Perl's hashing syntax and operations can be invoked using Perl's object-oriented syntax conventions.

```
$type = $obj->{Type};  
$obj->{Vendor} = "Cisco";  
$fan1 = $obj->findFan( 1 );
```

### 4 Close the session

As before - the session should be closed when no longer required.

```
$session->detach( );
```

## Using Primitives and Object References

The API provides function calls for accessing all the low-level facilities of CNCC Managers. Each of these primitives can be invoked with reference to the *InCharge::session* handle (see above), and takes arguments that exactly match the CNCC API's syntax.

The API also provides an object-oriented abstraction layer that allows Perl code to access the CNCC Manager using a syntax that is very similar to ASL. For example, in ASL you can list the vendors of all routers using this logic.

```
routers = getInstances( "Router" );  
foreach router ( routers ) {  
    obj = object( "Router", router );  
    vendor = obj->Vendor;  
    print( router . " - " . vendor );  
}
```

When using the API to perform the same action, the code looks like this.

```
@routers = $session->getInstances( "Router" );  
foreach $router ( @routers ) {  
    $obj = $session->object( "Router", $router );  
    $vendor = $obj->{Vendor};  
    print $router . " - " . $vendor . "\n";  
}
```

The two code fragments above are very similar. The first main difference is a matter of syntax. Perl uses ``\$" and ``@" to denote scalar and array variables, and {..} to denote object properties, which are hash table lookups. The second difference is that the *object( .. )* and *getInstances( .. )* functions are called with reference to a session handle in the Perl code.

## Event Subscription

The CNCC Perl API provides mechanisms for subscribing to and acting upon events generated by CNCC Servers.

The programming model delivers two different modes of CNCC Client/CNCC Server communication. The most direct is where a CNCC Client makes a request of the CNCC Manager which actions the request and responds. A simple example of this is an object query or update. For example, the action of obtaining the vendor of a particular device is one such query. In Perl, this query would be encoded in a manner similar to the following fragment.

```
use InCharge::session; use InCharge::object;
$session = InCharge::session->init( );
$device = "Router::gw1";
$obj = $session->object( $device );
$vendor = $obj->{Vendor};
print $vendor . "\n";
```

The second mechanism provides asynchronous notifications via subscriptions and is used when the CNCC Client program needs to listen for events generated by the CNCC Manager in response to other external events. One example would be a script that waits for the Vendor field of a particular router to change. In Perl, this could be coded in the following way.

```
use InCharge::session; use InCharge::object;
$session = InCharge::session->init( );
$observer = $session->observer( );
$device = "Router::gw1";
$session->propertySubscribe($device, "Vendor", 30 );
while ( 1 ) {
    @event = $observer->receiveEvent( );
    print "Vendor $event[2]::$event[3] is now \
                                                $event[5]\n";
}
```

The following sections provide an overview of mechanisms for creating and controlling a number of different types of subscriptions.

## Registering an Observer

In order to allow CNCC Servers to send subscribed events to a CNCC Client program, the CNCC Client must first register itself with the CNCC Manager as an event observer.

In Perl, this is done using the *observer()* method of the *InCharge::session* module. There are two steps needed to perform this action.

First, the CNCC Client must connect to the CNCC Manager, establishing a new *InCharge::session* connection. This is done using either the *InCharge::session->new* or *InCharge::session->init* methods, depending on the level of control that the script requires to impose on the connection process. The result of this step, whichever procedure call is used, is a valid *InCharge::session* object handle.

Here are some example code fragments that achieve this goal.

```
$session = InCharge::Session->init( )
```

This parses the script command line for the --server, --broker and other standard options.

```
$session = InCharge::session->new(  
    broker=>"192.168.0.3",  
    domain=>"InCharge"  
);
```

This allows the connection options to be specified explicitly by the script itself.

Once the script has obtained a handle that references the script/CNCC Server connection, *\$session* in the examples above, it can be used to obtain a second connection to the notification engine of the CNCC Manager. This is obtained using the *observer()* method on the *InCharge::session* handle just obtained. The following code performs this action.

```
$observer = $session->observer();
```

This *\$observer* handle now references a second link between the CNCC Client script and the CNCC Manager, which is used to pass subscribed events to the CNCC Client in real-time.

The observer connection can be closed, detached, or destroyed using the *detach()* object method:

```
$observer->detach();
```

This action has the side effect of cancelling any outstanding subscriptions.

# Subscribing to Notifications.

Once the CNCC Client has registered itself as an observer, the next step is to inform the CNCC Manager about which events it is interested in receiving. CNCC allows CNCC Clients to subscribe to a number of different types of events. These are listed in the following table.

METHOD TYPE	DESCRIPTION	METHOD API CALL
property	Notifications about changes to specified object properties in the ICIM database. For example, when the “Vendor” field of Router::gw1 changes	propertySubscribe propertyUnsubscribe
topology	Notifications about changes to the topology, such as the creation and deletion of objects. This does not refer to object property changes.	topologySubscribe topologyUnsubscribe
event	Notifications about the posting and clearing of events and changes to their state.	subscribe unsubscribe subscribeAll unsubscribeAll getSubscriptionState IsSubscribed

Table 3: **Subscription Methods Summary**

This tables gives the names of the methods used to subscribe to and unsubscribe from different types of notifications.

The following code segment is an example script that subscribes to changes of the Vendor field of every device in the topology.

```
$session = InCharge::session->init( );
$obs = $session->observer();
foreach $name ( $session->getClassInstances(
    "ICIM_UnitaryComputerSystem" ) ) {
    $session->propertySubscribe("::$name", "Vendor", 30);
}
```

# Receiving Notifications

Once the script has registered as an observer, and subscribed to the notifications of interest it then proceeds to listen for events and process them as required. The event reception method call is *receiveEvent()*. This returns an array of up to five values.

For the purposes of the descriptions that follow, we assume that events are returned in the array `@event`, as shown in the following script fragment.

```
@event = $observer->receiveEvent( );
```

Should the script require the event to be a single string with a separator used to delimit the fields, in the style of the ASL language, then the application can use the standard Perl join function:

```
$fs = "|";
$event = join( $fs, $observer->receiveEvent( ) );
```

The `receiveEvent()` method can take an optional parameter to specify a time-out in seconds, which may be fractional. If no event arrives within the specified time, a pseudo-event of type `TIMEOUT` is returned. For example,

```
@event = $observer->receiveEvent( 0.25 );
```

If no time-out is specified, the call waits forever.

The first element of the `@event` array, accessed using the Perl syntax: `$event[0]`, contains the event's time stamp measured using normal unix `time_t` semantics; i.e., the number of seconds since midnight January 1st, 1970.

The second element of the `@event` array, `$event[1]`, contains a text string that describes the type of event received.

The array elements from `$event[2]` to `$event[$#event]` have meanings that depend on the semantics of the event type given in `$event[1]`.

#### Event notification records

Event notifications are received from the CNCC Manager when the status of an event changes. The format of the notification record is:

EVENT RECORD ENTRY	DESCRIPTION
<code>\$event[0]</code>	Timestamp (INTEGER)
<code>\$event[1]</code>	“NOTIFY”
<code>\$event[2]</code>	Object class name (STRING)
<code>\$event[3]</code>	Object instance name (STRING)
<code>\$event[4]</code>	Event name (STRING)
<code>\$event[5]</code>	Event certainty (FLOAT)

Table 4: **Notification Record - NOTIFY**

Normally, the CNCC Manager sends a single notification message when an event becomes active and a single clear message when the event is no longer active. If an event corresponds to a root cause problem, it is possible that the certainty of the diagnosis will change over time. If the diagnosis certainty changes, the CNCC Manager generates another notification. Notifications of this type are streamed in a slightly different manner. This difference in behavior is a feature of the front end Perl API, not the CNCC Manager. The CNCC Manager sends NOTIFY messages in both cases. The API keeps internal notes about active events, and changes the event type accordingly.

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	“CERTAINTY_CHANGE”
\$event[2]	Object class name (STRING)
\$event[3]	Object instance name (STRING)
\$event[4]	Event name (STRING)
\$event[5]	Event certainty (FLOAT)

**Table 5: Notification Record - CERTAINTY\_CHANGE**

When an event is cleared by the CNCC Manager, the format of the record is:

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	“CLEAR”
\$event[2]	Object class name (STRING)
\$event[3]	Object instance name (STRING)
\$event[4]	Event name (STRING)

**Table 6: Notification Record - CLEAR**



### Object Create/Delete Records

An object create message is sent by the CNCC Manager when a new object is created in the CNCC Manager's repository. The format of an object create record is:

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	"CREATE"
\$event[2]	Object class name (STRING)
\$event[3]	Object instance name (STRING)

**Table 7: Notification Record - CREATE**

An object delete message is sent by the CNCC Manager when an object is deleted from the CNCC Manager's repository. The format of an object delete record is:

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	"DELETE"
\$event[2]	Object class name (STRING)
\$event[3]	Object instance name (STRING)

**Table 8: Notification Record - DELETE**

### Class Load Records

A class load message is sent by the CNCC Manager when a new class is created in the CNCC Manager's repository. Classes are created when new MODEL-generated libraries are loaded. The format of a class load record is:

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	"CLASS_LOAD"
\$event[2]	Class name (STRING)

**Table 9: Notification Record - CLASS\_LOAD**

### Relation/Property Change Records

A relation change message is sent by the CNCC Manager when a relationship between objects changes. The format of a relation change record is:

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	“RELATION_CHANGE”
\$event[2]	Object class name (STRING)
\$event[3]	Object instance name (STRING)
\$event[4]	Relation name (STRING)

Table 10: **Notification Record - RELATION\_CHANGE**

A property change message is sent by the CNCC Manager when an object’s property changes. The format of a property change record is:

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	“ATTR_CHANGE”
\$event[2]	Object class name (STRING)
\$event[3]	Object instance name (STRING)
\$event[4]	Property name (STRING)

Table 11: **Notification Record - ATTR\_CHANGE**

### CNCC Manager Connect/Disconnect Records

A CNCC Manager disconnect record is generated when the connection to the CNCC Server is lost. This differs somewhat from ASL operation. See the *observer()* primitive for proper handling of DISCONNECT events if 'restartableServer' operation is desired. These records are generated even if no subscriptions to the CNCC Manager are issued. The format of the CNCC Manager disconnect message is:

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	"DISCONNECT"
\$event[2]	Domain name (STRING)

Table 12: **Notification Record - DISCONNECT**

**Note:** There is no CONNECT record. In ASL, these are an artifact of the restartableServer front end that the Perl API does not provide. The restartableServer affords a means of invisibly attempting a (re)connection; the CONNECT message is an indication of success. The Perl API instead gives an immediate error on failure of *InCharge::session-->init()* or similar. It remains for the developer to provide retry logic to successfully CONNECT.

### Subscription status records

When the CNCC Manager receives a subscription request, it normally sends a notification back to the CNCC Client to indicate whether or not the request was accepted. In the event of an error such as an invalid event name being specified, it does not report an error using normal Perl *die* semantics but uses a notification to report that the subscription was rejected. The format of the ACCEPT/REJECT message is:

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	“ACCEPT” or “REJECT” or “PROPERTY_ACCEPT” or “PROPERTY_REJECT”
\$event[2]	Object class name (STRING)
\$event[3]	Object instance name (STRING)
\$event[4]	Event or property name (STRING)

Table 13: **Notification Record - ACCEPT, REJECT, (PROPERTY)**

### Event suspension records

Under certain circumstances, the CNCC Manager will elect to suspend events if they are temporarily irrelevant. For example, when an aggregation contains no triggering events a SUSPEND message is sent to the subscribed CNCC Client. The format of the SUSPEND message is:

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	“SUSPEND” or “PROPERTY_SUSPEND”
\$event[2]	Object class name (STRING)
\$event[3]	Object instance name (STRING)
\$event[4]	Event or property name (STRING)
\$event[5]	Descriptive message (STRING)

Table 14: **Notification Record - SUSPEND (PROPERTY)**

### Time-out records

If no event arrives within the time specified as the optional argument to *receiveEvent()*, a TIMEOUT record is returned, whose message format is:

EVENT RECORD ENTRY	DESCRIPTION
\$event[0]	Timestamp (INTEGER)
\$event[1]	“TIMEOUT”
\$event[2]	Domain name (STRING)

Table 15: **Notification Record - TIMEOUT**



# Object

## Name

*InCharge::object* - Access to CNCC repository objects.

## Synopsis

```
use InCharge::session;
$obj = $session->object( $class, $instance );
$value = $obj->get( "PropertyName" );
$value = $obj->{PropertyName};
$obj->put( "PropertyName", $value );
$obj->{PropertyName} = $value;
$rtn = $obj->invoke( "OperationName", .. arguments .. );
$rtn = $obj->OperationName( .. arguments .. );
```

## Description

The *InCharge::object* module allows objects in the CNCC repository to be manipulated in an object-oriented style, similar to the ASL language.

An *InCharge::object* reference is required to access the properties or methods of a Manager object. This reference is returned from the *object()* or *create()* methods of the *InCharge::session* module. Both methods allow access to pre-existing objects but the *create()* method will also create the object if it does not already exist. While *create()* always accesses the CNCC Manager, *object()* may or may not, depending on the invocation technique.

```
$obj = $session->object( "Router", "edgert1" );
$obj = $session->object( "Router::edgert1" );
$obj = $session->create( "Router", "newrouter" );
```

---

**Note:** Whenever you specify the details of a repository instance to work with in the API you have a choice of two syntaxes. You can either specify the object class and instance names as individual arguments or you can run them together with a delimiting double-colon (::). The API handles these two forms identically.

---

If you don't know the class to which an object belongs, you can either use a class argument of *undef*, or a string with nothing before the double-colon (::). For example,

```
$obj = $session->object( undef, "edgert1" );
$obj = $session->object( "::edgert1" );
```

The option of omitting the class name does not work with the *InCharge::session->create()* method because CNCC cannot create an object without knowing which class to use. It does work with *InCharge::session->object()* and related calls because the process of referring to an existing instance can legitimately include a query to identify the object's class. Note that the API does additional work to determine the object's class, if you choose not to provide the class name in these calls, imposing a slight performance penalty.

Once an object reference has been created, it can be used to invoke the object's operations or access its properties. Access to an object's attributes or properties can be obtained using calls shown in the following example.

```
$vendor = $obj->get( "Vendor" );
$vendor = $obj->{Vendor};
($vendor,$model) = $obj->get( "Vendor", "Model" );
%properties = $obj->get( );
$obj->put( "Vendor", "Cisco" );
$obj->{Vendor} = "Cisco";
$obj->put( Vendor => "Cisco", Model => "2010" );
```

These examples show that object properties can be accessed using either the *get()* and *put()* methods or via the psuedo-hash syntax. The latter syntax is preferred because it is closer to the original CNCC built in ASL language logic.



Two special internal properties can be accessed using the hash syntax only. These give the name of the class and instance to which the object reference refers. Treat them as read-only fields.

```
$obj->{_class}           BUT NOT: $obj->get("_class")
$obj->{_instance}        BUT NOT: $obj->get("_instance")
```

Object operations can be invoked using the *invoke()* method, or directly, as in the example:

```
@ports = $obj->invoke( "findPorts" );
@ports = $obj->findPorts();
$port = $obj->
    invoke( "makePort", "1.0", "PORT-rtl/1.0", "Port" );
$port = $obj->makePort( "1.0", "PORT-rtl/1.0", "Port" );
```

Again, the latter syntax, calling the operation directly, is preferred.

Use the *invoke()* method to access an object operation that duplicates the name of any of the built-in methods of the *InCharge::object* class. The first of these two calls the *new()* operation of the object in the repository, whereas the second calls the built-in *new()* method of the *InCharge::object* class.

```
$obj->invoke( "new", "qtable" );
$obj->new( "qtable" );
```

Note that *InCharge::object* is used for accessing ICIM instance operations and properties only. If you wish to make other ICIM calls that refer to instances, such as *subscribe()*, use the features of *InCharge::session* directly. It is invalid to code:

```
$obj->propertySubscribe( "Vendor" );
```

Instead, use:

```
$session->propertySubscribe($class, $instance, "Vendor");
```

or,

```
$session->propertySubscribe( $obj, ``Vendor`` );
```

This is because the *propertySubscribe* is not a repository class operation but a primitive.

```
dmctl -s DOMAIN getOperations CLASSNAME | more
```

Likewise, to determine what properties can be accessed using this module use:

```
dmctl -s DOMAIN getProperties CLASSNAME | more
```

# Functions and Methods

## object

```
$object = $session->object( $class, $instance );
```

creates a new object reference.

## get

```
$result = $obj->get( $property_name [, $property_name ...] )
```

or,

```
$result = $obj->{$property_name};
```

or,

```
%all_properties = $obj->get( )
```

Gets the value for the specified `property(s)` of the object.

The type of return value depends on the calling syntax used *get()* or *hash()* and the Perl evaluation context, scalar or array, as shown in the following table.

EXPRESSION SYNTAX	PROPERTY TYPE	RETURN TYPE IN SCALAR CONTEXT	RETURN TYPE IN ARRAY CONTEXT
<code>\$obj-&gt;{prop}</code>	scalar	scalar	scalar in [0]
<code>\$obj-&gt;{prop}</code>	array	array ref	array ref in [0]
<code>\$obj-&gt;get("prop")</code>	scalar	scalar	scalar in [0]
<code>\$obj-&gt;get("prop")</code>	array	array ref	array

Table 16: **Return Type for Call Syntax and Perl Evaluation Context**

Multiple values are always returned in an array or array reference.

To access the content of a property whose name is held in a variable, use the Perl typical logic, as shown.

```
$propname = "Vendor";  
$value = $obj->{$propname};
```

You can also get multiple values in a single *get()* call by listing all the property names as arguments. The results are returned in an array. On a CNCC Manager version 6 or later, this is faster than using multiple single-property *get()* calls. On Servers before version 6; there is no difference.

```
( $vendor, $type ) = $obj->get( "Vendor", "Type" );
```

You can also call *get()* with no arguments, in which case it creates a hash containing all the object properties and relations. There is no syntactical advantage, but there is a significant speed advantage on CNCC Manager version 6 and later.

See also the *get\_t()* call which extends the functionality of this method by returning additional information, to identify the type of data held in the property.

---

**Note:** The *get()* and *get\_t()* primitives throw an error when used to access a nonexistent property or one that cannot be retrieved for any reason, whereas the psuedo-hash syntax simply returns an undef value. This difference allows the Data::Dumper logic to display an entire object without erring even when some properties cannot be retrieved.

---

## get\_t

```
( $type, $value ) = $obj->get_t( $property_name );  
or,  
@types_and_values =  
    $obj->get_t( $prop1 [, $prop2 [, $prop3 .. ] ] )  
or,  
%all_property_types_and_values = $obj->get_t( );
```

This is like the *get()* method, except that it returns the type of the return value as well as the value itself. The data types are encoded as integer numbers. If the return is an array, then *\$value* will receive a reference to the array. If the return is a scalar, then *\$value* will hold it. *\$session->TYPE()* can be used to convert the *\$type* integer value to a mnemonic string.

The second syntax gets the types and values for multiple properties. Each type/value pair is held in a 2-element sub-array within the returned data.

The third syntax gets the types and values for all the properties and relations of the object and stores them in a hash, indexed by the property names. This approach has a significant performance benefit when working with CNCC Manager version 6 and above.

Example:

```
$obj = $session->object( "Router:gw1" );  
( $type, $value ) = $obj->get_t( "Vendor" );  
print "Vendor value ='$value', \
```

```
type = ".$session>TYPE($type)."\n";
```

This example will print:

```
Vendor value='CISCO', type = STRING
```

## put

```
$object->put( $property_name, $value );
```

This method allows fields of the object to be modified in the CNCC repository.

This method is used in a number of ways, however, the use of the pseudo-hash syntax is the preferred option for syntactic equivalence to CNCC's native ASL language.

```
$obj->put( "Vendor", "Cisco" );  
$obj->{Vendor} = "Cisco";  
$obj->{ComposedOf} = [ ];
```

To set more than one property in a single call, use multiple name:value pairs, such as:

```
$obj->put( Vendor => "Cisco",  
          PrimaryOwnerContact => "Joe Bloggs" );
```

or

```
%updates = ( Vendor => "Cisco",  
             PrimaryOwnerContact => "Joe Bloggs" );  
$obj->put( %updates );
```

When using either syntax to set a relationship or list property, use a reference to a Perl array, such as:

```
$obj->{ComposedOf} = [ $a, $b, $c ];  
$obj->put( "ComposedOf", \@things );
```

Use *insertElement()* and *removeElement()* to add or remove elements from a list.

## isNull

```
$boolean = $object->isNull();
```

Tests to see whether the object is present in the repository.

TRUE means that the object is not present. FALSE means it is present.

## invoke

```
reply = $object->invoke( $operation, ... arguments ... );
```

Invokes the named repository class operation on the object.

The arguments passed should be as expected by the operation. If the operation returns a scalar value, the call should be called in a scalar context. If it returns an array, it should be invoked in an array context.

---

**Note:** The preferred way of achieving the same result is to use the operation name directly. Thus, the following are equivalent but the latter is preferred.

---

```
$obj->invoke( "makePort", "1.0", "First port", "Port" );  
$obj->makePort( "1.0", "First port", "Port" );
```

Also, see the *invoke\_t()* method, described below.

## invoke\_t

```
( $type, $value ) =  
    $object->invoke_t($operation, .. args .. )>
```

Invokes the named class operation on the object in the same way as *invoke()*, but *invoke\_t()* also returns the type of data returned by the call.

The data types are encoded as integer numbers. If the return is an array, then the *\$value* will receive a reference to the array. If the return is a scalar, then *\$value* will hold it.

## insertElement

```
$obj->insertElement( $relation, @object[s] );
```

Inserts the specified objects into an object relationship.

One or more can be specified to be inserted.

```
$obj->insertElement( "ComposedOf",  
    "Interface::IF-ether1",  
    "Interface::Loopback/0");  
$obj->insertElement( "ComposedOf", @interfaces );
```

## removeElement

```
$obj->removeElement( $relation, @item[s] );
```

Removes the specified items from an object relationship.

One or more items can be specified to be removed.

```
$obj->removeElement( "ComposedOf",  
    "Interface::IF-ether1",
```

```
                "Interface::Loopback/0" );  
$obj->removeElement( "ComposedOf", @interfaces );
```

### delete

```
$obj->delete( )
```

Deletes the specified item from the repository, but without performing any clean-up of inter-object dependencies.

Consider using the *remove()* operation, if one exists, instead for a more complete action.

### notify

```
$obj->notify( $event_name );
```

Notifies the specified event for the object.

```
$objref->notify( "Unresponsive" );
```

### clear

```
$obj->clear( $event_name );
```

Clear the specified event for the object.

```
$objref->clear( "Unresponsive" );
```

### countElements

```
$count = $obj->countElements( $relation )
```

Counts the number of elements in the given relationship or throws an error if \$relation is not a relationship.

```
$count = $obj->countElements( "ComposedOf" );
```

## Session

### Name

*InCharge::session* - CNCC Manager session management.

### Synopsis

```
use InCharge::session;
$session = InCharge::session->init( );
$session = InCharge::session->new( "InCharge" );
$session = InCharge::session->new(
    broker=>"localhost:426",
    domain=>"InCharge",
    username=>"noddy",
    password=>"bigears",
    traceServer => 1
);
$object = $session->object( "Host::toytown1" );
$object = $session->create( "Router::crossroads" );
(... and continuing with the methods, described below ...)
```

## Description

This module provides the mechanisms for accessing a CNCC Manager in a manner that is similar to that employed by CNCC's ASL scripting language. It provides the main access point to CNCC domains, allowing scripts to establish CNCC client/CNCC server connections and to obtain *InCharge::object* references which can be used to manipulate the objects in the topology.

Refer to the introductory chapter for an overview of this and the other *InCharge::\** modules and a simple tutorial description of how they are used.

## Function Groups

*InCharge::session* provides access to four kinds of functions.

### Session Management

Functions in this group are the principle functions of the module. They are used for managing the Perl CNCC client/CNCC Manager connection. Using these functions a script can attach, detach, listen for events, and create *InCharge::object* references.

### InCharge Primitives

The *InCharge::session* module permits access to the low-level primitive functions of the CNCC Manager, allowing actions such as *getClasses()* and *getInstances()* to be performed. These primitives do not all exactly mirror the interface provided by *dmctl* or the native ASL language. For example, *dmctl* has a *save* command that does not have an exact primitive equivalent, but there are two primitives that can be invoked to give the same results. These are *storeClassRepository()* and *storeAllRepository()*. Where primitives exist that semantically match *dmctl* or ASL commands but differ in name, aliased names are provided to give syntactic compatibility.

### Utility Functions

This group includes functions to provide additional logical assistance to writers of CNCC Perl scripts.



## Compatibility Functions

This group of functions provides wrappers around the primitives to provide an interface that is more consistent with CNCC's native ASL language and dmctl utility.

Wrapper functions of this type are only provided for functions where the syntax and semantics of the primitive is not compatible with ASL or dmctl. The save example has already been cited, above, to highlight one such function.

## Error Handling

Errors are reported back to the invoking script using Perl's *die* mechanism, and can be caught using *eval*. This is typical Perl coding practice and mimics the try-throw-catch logic of java and C++.

## Session Management Functions

The following session management functions are provided.

### **new**

```
$session = InCharge::session->new( .. options .. );
```

This function establishes a connection between the calling Perl script and a CNCC Manager and returns a tied reference that can be used thereafter to manipulate the domain and the entities contained in its repository. Possible options are:

**1** broker => \$host[:\$port]

This specifies the CNCC broker from which the domain details are to be lifted. The string consists of a host name or IP address followed by an optional port number, delimited by a colon.

The default host is localhost, and the default port is 426.

**2** domain => [\$host:\$port/]\$domain

**3** CNCC server => [\$host:\$port/]\$domain

This specifies the name of the domain to be used. If the host and port details are also given, then the API does not refer to the broker to determine them.

The default domain name is InCharge.

---

**Note:** The option name CNCC “server” can be used in place of “domain” and the two options have the same meaning.

---

4 user => \$user\_name

5 username => \$user\_name

This specifies the name of the user to be used in connecting to the domain. If user or username is specified, then password must also be specified. If the username is not given, then the API refers to the InChargeclientConnect.conf file to determine the authentication information to use when establishing the connection.

There is no default username.

If no username is specified, the script inspects and interprets the SM\_AUTHORITY environment variable in the same way that the main software does and may prompt the user for the user name and password using the standard I/O device.

6 password => \$password

This specifies the password for the user given with the username option.

---

**Note:** The username and password must both be supplied, or neither of them must be specified.

---

7 description => \$description

This describes the role of the script and is noted by the CNCC Manager for use in debug and other logging messages. Its contents are not significant, otherwise. The default is Perl-InCharge Client.

8 traceServer => 1

If specified and given a true value, non-zero, then InCharge server-level tracing is turned on. This causes the CNCC Manager to log information about every primitive call invoked by the script, which can quickly fill up the CNCC server's log file. It is recommended to use this sparingly since it also has a negative impact on the CNCC Manager's performance.

9 timeout => \$timeout

This specifies the timeout to be tolerated while waiting for responses from the CNCC Manager to primitive requests. The default value is 120 seconds. Take care not to make this value too low. Otherwise, slow-to-process requests will fail in a manner that looks like a communication link failure between the script and the CNCC Manager.

If only the domain name is given, it can be specified without the domain=> key.

The username and password fields are required if connecting to a SAM CNCC server or a CNCC Manager with authentication features enabled. If neither of these arguments is given, the `clientConnect.conf` file is used to determine the username and password or the mechanism to obtain them.

## init

```
$session = InCharge::session->init( );
```

This is the simplified version of `InCharge::session->new()`. It parses the script's command line, looking for options that specify the broker, CNCC Manager username, password, and trace options. Then, it invokes the primitive `InCharge::session->new( )` with those arguments and passes back the result.

`InCharge::session->init( )` looks for the following script command line arguments.

```
--broker=<brokerIP[:brokerPort]>      (also: -b)
--server=<domain-name>                 (also: -s)
--user=<username>                      (also: -u)
--password=<password>                 (also: -p)
--traceServer
--timeout
```

If neither the `--user` (or `-u`) and `--password` (or `-p`) are specified, the script makes use of the `SM_HOME/conf/InChargeclientConnect.conf` file to determine the username and password to be employed; see comments in the file for details of this mechanism. This mechanism is turned on by specifying the value `<STD>` for the `SM_AUTHORITY` environment variable.

If it encounters a command line syntax error, it calls `usageError`, in the main script, which the developer must provide. A single large text string containing a description of the standard options handled is passed as the argument to `usageError`, allowing the author to include information about the standard options as well as any non standard ones provided. If the `usageError` subroutine does not exist, a default error message is printed on `STDERR`.

Note that the `init()` function consumes (removes) the command line arguments it handles from `@ARGV` as it processes them; therefore you can access the `@ARGV` array after its execution to process additional arguments without needing to skip the standard ones. However, you cannot use the `init` command twice in the same script without saving and restoring the contents `@ARGV` first, as in the example:

```
@SAVE = @ARGV;
$session1 = InCharge::session->init( );
@ARGV = @SAVE;
```

```
$session2 = InCharge::session->init( );
```

## broken

```
$flag = $session->broken( );
```

Returns non-zero (TRUE) if the session with the CNCC Manager is broken in some way.

This indicates connection or protocol failures. To continue working with a broken session, the script should call the *reattach()* function, and then re-establish the event subscription profiles required.

## reattach

```
$session->reattach( );
```

Re-establishes a connection that has been detached or broken.

This can be called to reconnect to a CNCC server to which the connection has been lost. Re-establishing the connection does not automatically re-establish observer sessions, subscriptions, transactional or other session state information.

If the call is used to reattach a session which had an active observer, the observer connection is closed as a side-effect of the action and must be re-opened separately.

This function should be called after a [13] I/O Error is thrown by any of the CNCC Manager access calls in order to shut down and reopen the socket, cleaning up the protocol. If this step is not taken there is a danger that residual packets on the connection would cause synchronization problems between the CNCC client and CNCC Manager. See the Primitives/ErrorHandling discussion for an explanation of the [13] prefix.

---

**Note:** The *reattach()* primitive doesn't return a new session identifier but refreshes the referenced one. This is not a *dup()* style of action.

---

## detach

```
$session->detach( );
```

Detaches from the CNCC domain referred to by *\$session*.

This call can be used for either a session, created using *InCharge::session->new()*, or an observer session, created using *InCharge::session->observer()*.

If this is used to detach a session with an active observer, the observer is also closed.

This call does not completely destroy the *\$session* reference contents but retains enough information to allow the session to be re-established. Thus, it is possible to call *\$session->reattach( )* to re-connect to the CNCC Manager using the same parameters as were used in the initial connection. However, the event subscriptions need to be re-established explicitly in this event.

## observer

```
$observer_session = $session->observer( .. options .. );
```

Creates and returns a reference to a connection to a CNCC Manager from which subscribed events can be receive.

This establishes a new socket between the CNCC client and CNCC Manager. Once connected, events can be subscribed to using the various subscribe methods and they can be received using:

```
@event_info = $observer_session->receiveEvent( );
```

Specifying the option *connectEvents => 1* to the *observer()* function causes CNCC server disconnection to be notified as a DISCONNECT event rather than a [13] I/O Error. However, unlike ASL, the re-connection is not performed automatically. The script can use the *\$session->reattach( )* call to attempt an explicit re-connection and must then re-establish any event subscriptions and other contexts.

Specifying the option *ignoreOld => 1* causes events generated before the connection was established to be discarded automatically. The use of this option is not generally recommended since the atomicity of time measurement on NT and Unix makes its results somewhat unpredictable.

Repeated calls to the *observer( )* method of a session return references to the same observer. It is not possible to create multiple observers on the same session.

## receiveEvent

```
@event = $observer_session->receiveEvent( [ $timeout ] );
```

Listen for subscribed events from the CNCC Manager.

The received events are returned as an array or, in scalar context, a reference to an array containing three or more elements. The details of the significance of the different events are described in the subscriptions section.

The first element of all events is the time stamp, on the CNCC Manager's system clock, and not the CNCC client's clock. The second element is a string defining the event type. The other elements are event specific.

The `$timeout` is optional, and specifies a timeout period, in seconds, that the script is prepared to wait for an incoming event. If no event arrives in this time period, an event of type `TIMEOUT` is returned. The `$timeout` can be specified in fractions of a second, or “float”; e.g., 0.25 = a quarter second.

## object

```
$obj = $session->object( $objectname );
```

Creates a new *InCharge::object* reference that can be used to invoke methods of the *InCharge::object* module.

As an example, to obtain the value of the `Vendor` field for a particular object, use:

```
$obj = $session->object( "::gw1" );  
$vendor = $obj->{Vendor};
```

You can even combine these into a single line, such as:

```
$vendor = $session->object( "::gw1" )->{Vendor};
```

The *\$objectname* parameter can be specified in any of the following styles.

**1** `object( 'Router::gw1' )`

A single string where both the class and instance names are specified with double colons (`::`) delimiting them. If variables are to be used to specify the relevant parts of the string, then it is important that at least the variable before double-colon (`::`) is encased in braces because without them, Perl will give the (`::`) characters its own meaning.

**2** `object( 'Router', 'gw1' )`

An example of two strings with one for the class and one for the instance name.

**3** `object( '::gw1' )`

An example of one string with the class name missing. The API will make a query to the CNCC Manager to discover the actual class for the object causing a minor performance penalty.

**4** `object( undef, 'gw1' )`

An example of two parameters with the first one undefined. This also results in the API performing a CNCC Manager query.

**5** `object( 'gw1' )`

An example of a single parameter that doesn't include the double-colon (::) delimiter, which must contain just the instance name. As above, a CNCC Manager query is performed to determine the relevant class name.

An important difference between the API and the native ASL language is that if you create an object, using *object()*, in native ASL without specifying the class name, the language assumes that the class *MR\_Object* can be applied. This restricts the level of property and operation access that can be used. The API queries the repository to determine the actual class for the instance, giving complete access to the resulting object's features.

## create

```
$obj = $session->create( $objectname );
```

Like *object()* above, the *create()* call creates an *InCharge::object* valid reference through which a specified instance can be manipulated. However, unlike *object()*, above, the *create()* method creates the object if it doesn't already exist.

Since it has the ability to create objects, it is important that the object name specified as an argument includes both the instance name and the class name. You cannot use the *::instance* or *(undef, \$instance)* syntaxes for specifying the object name. You can, however, use either the *Class::Instance* or *(\$class, \$instance)* syntax described for the *object()* method above.

Unlike the *createInstance()* primitive, it is not an error to call the *create()* method for an object instance that already exists. In this case, the call is equivalent to the *\$session->object()* call above and it simply returns the *InCharge::object* valid reference to the instance.

## callPrimitive

```
RESULT = $session->callPrimitive($primitiveName, @arguments)
```

Calls the specified CNCC Manager primitive, passing it the arguments and returning its result.

---

**Note:** For most primitives, this is a complex invocation sequence. However, it is only actually needed when a primitive and a method of the *InCharge::session* module share the same name, and you wish to use the primitive version.

---

The following are equivalent, although the first is preferred.

```
@list = $session->getInstances( "Router" );  
@list =  
    $session->callPrimitive("getInstances", "Router");
```

The *put()* primitive is one of the few primitives where these two ways of calling it are not equivalent. This is because the *InCharge::session* module exports its own variant of the method. If you must gain access to the primitive version, you will need to use the *callPrimitive()* mechanism. However, this is not recommended, since the syntax is complex. See the documentation on the *put()* method for more details.

The type of the RESULT in array or scalar context is dependant on the primitive being called. In general, if the primitive returns a scalar you get a scalar or, in array context, a single element array. If the primitive returns an array you get an array, in array context, or array reference, in scalar context.

## Utility Functions

### TYPE Function

```
$number = $session->TYPE( $string );  
$string = $session->TYPE( $number );
```

Converts a CNCC Manager data type mnemonic string to its internal numeric code, or vice versa. So the following prints “13”.

```
print $session->TYPE( "STRING" ) . "\n";
```

The following code prints “STRING”.

```
print $session->TYPE( 13 ) . "\n";
```

### getFileno

```
$fno = $session->getFileno( );
```

Returns the underlying system file number the refers to the socket used for the script/CNCC server connection.

This is useful when the script wishes to use the Perl *select* statement to listen for events from multiple domain CNCC servers, using multiple observer objects.

### getProtocolVersion

```
$ver = $session->getProtocolVersion( );
```

Returns the protocol version number supported by the CNCC Manager.



This is a single integer number derived by the following calculation.

$$( \text{major} * 10000 ) + ( \text{minor} + 100 ) + \text{revision}$$

Hence, version “V5.1” is represented by the number 50100, and version “V4.2.1” is represented by 40201.

## primitiveIsAvailable

```
$boolean = $session->primitiveIsAvailable( $primitive_name )
```

Checks whether the named primitive is available in the CNCC Manager.

A value of 1 means that it is available, and value of 0 means that it is not available, either because it is an undefined primitive or was introduced in a later version of the CNCC Manager software.

```
if ( $session->primitiveIsAvailable (
    "getMultipleProperties" ) {
    $vendor, $model ) = $session->getMultipleProperties (
        $obj, [ "Vendor", "Model" ] );
} else {
    $vendor = $obj->{Vendor};
    $model = $obj->{Model};
}
```

# Compatibility Functions

The following functions add varying degrees of wrapper logic round the CNCC primitives, to make them more compatible with the native ASL language.

## save

```
$session->save( $filename [, $class ] );
```

Saves the repository in the specified file.

If a class name is specified then only the instances of that class are saved.

## put

```
$session->put( $object, $property, $value );
```

It is not recommended that this method be used extensively. Instead, use the features of *InCharge::object*.

This method changes the value of an object property. This version differs from the *put\_P()* primitive in that the latter requires the value type to be specified explicitly, whereas this version determines and caches the type. The following calls are, therefore, equivalent, although the first is preferred.

```
$obj = $session->object( "Router::gw1" );
$session->{Vendor} = "Cadbury";
$obj->put( "Vendor", "Cadbury" );
$obj->put( Vendor => "Cadbury" );
$session->put( "Router::gw1", "Vendor", "Cadbury" );
$session->object( "Router::gw1" )->{Vendor} = "Cadbury";
$session->callPrimitive( "put_P", "Router", "gw1",
                        "Vendor", [ "STRING", "Cadbury" ] );
```

When giving a value to an array property, such as the *ComposedOf* relationship, pass an array reference as shown in the example:

```
$obj->{ComposedOf} = [
    "Interface::IF-if1",
    "Interface::IF-if2"
];
```

Also, you can set more than one property in a single call. This can reduce complexity in the script layout but has minimal performance advantage.

```
$obj->put (
    Vendor    => "CISCO",
    Model     => "2500",
    Location  => "Behind the coffee machine"
);
```

## invoke

```
RESULT = $session->invoke($object, $operation[, @arguments]);
```

It is not recommended that this method be used extensively. Instead, use the features of *InCharge::object*.

This method invokes the specified object operation, passing it the listed arguments and returning the *RESULT*.

The type of the *RESULT* in array or scalar context is dependant on the operation being called. In general, if it returns a scalar you get a scalar or, in array context, a single element array. If it returns an array you get an array, in array context, or array reference, in scalar context.

---

**Note:** This method's semantics and syntax differ from the primitive method *invokeOperation()* in that the latter needs to have the types of the arguments specified explicitly, whereas for this method the *InCharge::session* module version discovers and caches the operation argument types and does not require the arguments to be listed in arrays of array references.

---

Additional documentation about the operations that exist for a particular class can be obtained using the *dmctl* utility, as shown:

```
dmctl -s DOMAIN getOperations CLASSNAME
```

The following examples are equivalent; the first example is preferred.

```
$obj = $session->object( "Router::gw1" );
$fan= $obj->findFan( 2 );
$fan = $session->invoke( "Router::gw1", "findFan", 2 );
$fan = $session->callPrimitive( "invokeOperation",
    "Router", "gw1", "findFan",
    [ [ "INT", 2 ] ]
);
```

## invoke\_t

```
($type, $value) =
    $session->invoke_t( $object, $operation [, @arguments]
);
```

Identical to *invoke()* except that the return indicates both the type and the value of the returned data.

The value is a Perl scalar, if the operation returns a scalar, or an array reference, if the operation returns an array. The type will contain one of the CNCC Manager internal type codes. For example, “13” is the code for a string.

## findInstances

```
@instances =  
    $session->findInstances( $c_patn, $i_patn [, $flags] )  
  
or  
  
@instances =  
    $session->findInstances( "${c_patn}::${i_patn}"  
                            [, $flags] )
```

Finds instances that match the class and instance patterns, according to rules specified in the flags.

The \$flags is a set of characters that modifies the way the call works.

A flag of “n” means that subclasses are not recursed into. Therefore, instances in matching classes only are returned. Without “n”, instances of matching classes and their subclasses are returned.

A flag of “r” means that unix-like RegEx matching is used during the search. If the “r” flag is not specified, the search uses CNCC glob pattern matching.

The default is no flags; i.e., glob matches and recursion.

Results are returned as a list of strings, each of which contains a class and instance name delimited with double-colon (::).

---

**Note:** The search strings are anchored as if the “^” and “\$” had been used in the unix-style pattern. Therefore, “rr\*” matches “rred” but not “herring”, whereas “\*rr\*” matches both of them.

---

Example:

```
@found = $session->findInstances( "Router:gw*", "n" );
```

## getCauses

```
@events =  
    $session->getCauses( $objectname, $event [, $oneHop] );
```

The *getCauses()* function returns a list of problems that cause an event.

The function receives arguments class, instance (possibly combined into one), and event. The function returns the problems that cause the event based on the relationships among instances defined in the CNCC Manager.

The `oneHop` parameter is optional. If it is omitted or passed as `FALSE`, the full list of problems explaining `eventname`, whether directly or indirectly, is returned. If it is passed as `TRUE`, only those problems that directly list `eventname` among the events they explain are returned.

The function returns an array of array references with the format:

```
[
  [ <classname::instancename>,<problemname> ],
  [ <classname::instancename>,<problemname> ],
  ...
]
```

---

**Note:** The class and instance names are returned as a single double-colon (::) delimited string, giving two strings per returned event in total. This is different from the native ASL language which returns the class and instance names separately, giving three strings for each event.

---

Example:

```
@causes =
    $session->getCauses( "Router::gw1",
                        "MightBeUnavailable"
    );
```

## getClosure

```
@events =
    $session->getClosure($object, $eventname[, $oneHop]);
```

The `getClosure()` function returns a list of symptoms associated with a problem or aggregation.

The function returns the symptoms associated with the problem or aggregate based on the relationships among instances defined in the CNCC Manager.

The `oneHop` parameter is optional. If it is omitted or passed as `FALSE`, the full list of problems explaining `eventname`, whether directly or indirectly, is returned. If it is passed as `TRUE`, only those problems that directly list `eventname` among the events they explain are returned.

The function returns an array of array references with the format:

```
[
  [ <classname::instancename>,<problemname> ],
  [ <classname::instancename>,<problemname> ],
  ...
]
```

---

**Note:** The class and instance names are returned as a single double-colon (::) delimited string, giving two strings per returned event in total. This is different from the native ASL language which returns the class and instance names separately, giving three strings for each event.

---

Example:

```
@symptoms =  
    $session->getClosure( "Router::gw1", "Down", 0 );
```

## getExplains

```
@events =  
    $session->getExplains($object, $eventname[, $onehop ]);
```

MODEL developers can add information to a problem in order to emphasize events that occur because of a problem. The *getExplains()* function returns a list of these events.

The \$onehop parameter is optional. If it is omitted or passed as FALSE, the full list of problems explaining \$eventname, whether directly or indirectly, is returned. If it is passed as TRUE, only those problems that directly list eventname among the events they explain are returned.

The function returns an array of array references with the format:

```
[  
    [ <classname::instancename>,<problemname> ],  
    [ <classname::instancename>,<problemname> ],  
    ...  
]
```

---

**Note:** The class and instance names are returned as a single double-colon (::) delimited string, giving two strings per returned event in total. This is different from the native ASL language which returns the class and instance names separately, giving three strings for each event.

---

## getExplainedBy

```
@events =  
    $session->getExplainedBy($object, $event[, $onehop ]);
```

This function is the inverse of the *getExplains()* function.

It returns those problems which the MODEL developer has listed as explaining this problem.

The \$onehop parameter is optional. If it is omitted or passed as FALSE, the full list of problems explaining \$event, whether directly or indirectly, is returned. If it is passed as TRUE, only those problems that directly list \$event among the events they explain are returned.

The function returns an array of array references with the format:

```
[
    [ <classname::instancename>,<problemname> ],
    [ <classname::instancename>,<problemname> ],
    ...
]
```

**Note:** The class and instance names are returned as a single double-colon (::) delimited string, giving two strings per returned event in total. This is different from the native ASL language which returns the class and instance names separately, giving three strings for each event.

subscribe and unsubscribe

```
$session->subscribe( $C, $I, $E [, $flags ] );
$session->subscribe( "$C::$I::$E[/$flags]" );
$session->unsubscribe( $C, $I, $E [, $flags ] );
$session->unsubscribe( "$C::$I::$E[/$flags]" );
```

These functions subscribe, or unsubscribe, to notifications of the specified events. “\$C”, “\$I”, “\$E” must be regexp patterns representing the classes, instances, and events to which to subscribe.

The *unsubscribe()* function is the inverse of *subscribe()*.

The \$flags value is a bitwise combination of the values in the following table or a more mnemonic string, as shown in the table.

FLAG BITFIELD VALUE	DESCRIPTION
0x000001	Simple event
0x000002	Simple aggregation
0x000010	Problem
0x000020	Imported event
0x000040	Propagated aggregation
0x0000ff	All
0x001000	Expand subclasses

FLAG BITFIELD VALUE	DESCRIPTION
0x002000	Expand subclasses events
0x004000	Expand aggregations
0x008000	Expand closures
0x010000	Sticky
0x020000	Undo all
0x040000	Quiet accept
0x080000	Quiet suspend
0x100000	Glob

Table 17: **Subscription Flag Parameter Values**

As a compatibility aid, the `$flag` can also be specified as a string of letters. In this case, each of the letters are subscription qualifiers: 'p' means subscribe to problems, "a" means subscribe to aggregates (impacts), and "e" means subscribe to events. If none of these are present, "p" is assumed. The letter "v" means run in verbose mode, which turns on subscription control messages. The action of these options is the same as that provided by the `sm_adapter` program's `--subscribe=` option.

Examples:

```
$session->subscribe( "Router", ".*", ".*", "/pev" );
$session->subscribe( "Router::.*::*/peav" );
$session->subscribe( $obj, ".*", 0x3 );
$session->unsubscribe( $obj, ".*", 0x3 );
```

## transaction, abortTxn and commitTxn

```
$session->transaction( [ $flag ] );
$session->abortTxn( );
$session->commitTxn( );
```

Transactions, Commit and Abort.

Using transactions, you can commit many changes to the objects in a CNCC Manager as a single atomic "transaction" or choose to abort all of them. Use the following syntax to create a transaction:

```
$session->transaction();
```



After initiating the transaction, every change made to an object does not affect the object until you commit the transaction. If the transaction is aborted, any changes made will not affect the object. Use the following syntax to either commit or abort a transaction.

```
$session->commitTxn( );
```

or

```
$session->abortTxn( );
```

The changes made with a transaction are not visible outside of the script until the changes are committed. Within a transaction, the same script can see the proposed changes. Transactions also can control how other applications see objects before changes are committed or aborted by adding a single keyword. The syntax of a transaction with a keyword is:

```
$session->
    transaction([ "WRITE_LOCK" | "READ_LOCK" | "NO_LOCK" ] );
```

A keyword can be any one of the following:

KEYWORD	DESCRIPTION
WRITE_LOCK	While the transaction is open, no other process can modify or access information in the repository.
READ_LOCK	Currently behaves the same as WRITE_LOCK
NO_LOCK	This is the default behavior. No locks exist until the script commits the transaction.

Table 18: **Transaction Lock Options**

Transactions may be nested. When you nest a transaction, you must commit or abort the nested transaction before you commit or abort the previous transaction.

The API aborts any open transactions when the script terminates.

Example:

```
#!/usr/local/bin/Perl
$session = InCharge::session->init( );
$delthis = shift @ARGV;
$delthisObj = $session->object($delthis);
@relObj = @{ $delthisObj->{ComposedOf} };
$session->transaction();
$x = $delthisObj->delete();
foreach $mem (@relObj) {
    $mem->delete();
}
```

```
$session->commitTxn();  
print("Deleted ".delthis." and related ports\n");
```

In the example, the script deletes a card and its related ports. The script is invoked with an argument that specifies the card to delete. Using the `ComposedOf` relationship, the script creates a list of Port objects to delete. The script deletes the card and its related ports at the same time through a transaction ensuring that no other script can see the intermediate stage with an incompletely deleted suite of objects.

# delete

```
$session->delete( $object );
```

Deletes the specified object instance from the repository.

---

**Note:** This does not clean up all the object inter-dependencies and links. For a cleaner object deletion, use the *remove()* operation, if one exists, for the object class in question; see the *invoke()* primitive.

---

The *delete()* method can be called in one of two ways.

```
$session->delete( $object );
```

or

```
$object->delete();
```

# getEventType

```
$type = $session->getEventType( $class, $event );
```

Given a class and event name, this call returns a string that describes the type of the event. The possible strings returned are:

EVENT TYPE LITERAL	DESCRIPTION
EVENT	Event
AGGREGATION	Aggregation
SYMPTOM	Symptom
PROBLEM	Problem
UNKNOWN	Error indication

Table 19: **Event Types**

Example:

```
$type = $session->getEventType( "Router", "Down" );
```

To obtain the low-level numeric type codes, instead of descriptive strings, use the *getEventType()* primitive, as shown.

```
$type =  
$session->primitive( "getEventType", "Router", "Down" );
```

## getServerName

```
$session->getServerName( );
```

Returns the name of the CNCC Manager to which the session is connected.

## insertElement

```
$session->insertElement( $object, $relation, @item[s] );
```

Inserts one or more elements into an object relationship.

It is suggested that the *insertElement()* feature of the *InCharge::object* module be used instead, as shown.

```
$obj->insertElement( $relation, @item[s] );
```

## removeElement

```
$session->removeElement( $object, $relation, @item[s] );
```

Removes one or more elements from an object relationship, such as *ComposedOf*.

It is recommended that the *removeElement()* feature of the *InCharge::object* module be used instead, as shown.

```
$obj->removeElement( $relation, @item[s] );
```



# Primitives

## Name

InCharge::primitives - Low-level primitive calls to CNCC

## Description

This section lists and describes the CNCC Manager primitives, the low-level remote calls, that are supported by the API. These primitives provide the standard protocol between client applications, such as dmctl, ASL adapters, API scripts, and the CNCC Console and the CNCC Manager. This manual page describes the Perl syntax for calling these primitives and provides outline documentation on their use and semantics.

## Primitive Naming Conventions

The names given to the primitives follow a convention of using lower case, except for the first letter of the second and subsequent words of multi-word names. For example, to get operation arguments, we use the name `getOperationArguments`.

Where the resulting names are overly long, the API provides shorter aliases; *getOperationArguments()* has the alias *getOpArgs()*. Typically, the word “Operation” is shortened to “Op”, and “Property” is shortened to “Prop”, however, both the long and shortened name can be used. Both forms are listed here.

Since primitives are designed to be called via the *InCharge::session*, where a primitive name conflicts with a module function, the name of the primitive has the string “\_P” concatenated onto it in order to differentiate the two. Script authors are discouraged from using these “\_P” versions since higher level versions are available through *InCharge::session* and, in some cases, *InCharge::object* that are easier to use.

Where a primitive returns a value that may be of any type, a second version of the call is provided that returns both the numeric type code and the return value. The name of this extended version is the same as the lesser original but with “\_t” appended. You can also specify “\_T” instead of “\_t”, in which case when the primitive returns an ANYVAL\_ARRAY\_SET (i.e., a structure of structures), the fields of the structures are also accompanied by their types. This is a reference to a 2-element array containing type and value is given for each structure field.

The primitive names are similar to those used in the C++ API. Where the names do not match those used by ASL or dmctl, aliases are provided. For example, the ASL command *getInstances()* is called *getLeafInstances()* in the C++ API, therefore, the API allows both names to be used. The C++ name is the name used for the actual primitive and the ASL name is provided as an alias.

The C interface for CNCC, on the other hand, uses function names that look like *sm\_property\_unsubscribe()*. They start with “sm\_” and use all lower case words delimited by underscores. This set of functions is less complete than the C++ equivalent interface and does not provide a one-to-one match of all the CNCC Manager primitives. The API for Perl does not provide a match for the C interface function names.

## Primitive Calling Conventions

All the functions described in this document must be invoked with reference to a valid object of the *InCharge::session* module. These object references are created using *InCharge::session->object()*, *InCharge::session->create()*, or *InCharge::session->getInstances()*.

The general approach used is as follows.

- 1 Initialize a session and obtain a reference to it.

```
$session = InCharge::session->init( );
```

- 2 Call the primitives required using the session reference; e.g.,

```
foreach $class ( sort $session->getClasses() ) {  
    foreach $inst (  
        sort $session->getInstances($class))  
    {  
        print $class . ":@" . $inst . "\n";  
    }  
}
```

- 3 Close the session.

```
$session->detach( );
```

Where access to operations or properties of CNCC repository objects is required you are discouraged from using the *get()*, *put()* and *invokeOperation()* primitives, but encouraged to use the features of the *InCharge::object* module instead. Using this approach, the script obtains an *InCharge::object* reference, which is used to access the required information. For example,

- 1 Establish a session.

```
$session = InCharge::session->init();
```

- 2 Create an *InCharge::object* valid reference to the object of interest.

```
$obj = $session->object( "Router::gw1" );
```

- 3 Manipulate the object using the reference.

```
$type = $obj->{Type};  
$obj->{Vendor} = "Cisco";  
$fan1 = $obj->findFan( 1 );
```

- 4 Close the session.

```
$session->detach( );
```

## Error Handling

All the functions and methods of objects in the API modules throw errors using the Perl *die* command. In order to catch any errors that may occur, the *eval()* function can be used and the “\$@” variable inspected after the event. This is common Perl scripting practice.

The example shown in the following script will abort if the router “gw1” does not exist in the topology at the line where the name of the Vendor is queried, and the last line will not be executed.

```
use InCharge::session;
$session = InCharge::session->init();
$vendor = $session
    ->object( "Router::gw1" )
    ->get("Vendor");
print "Vendor is $vendor\n";
```

To trap this possible error, the code can be modified as follows.

```
use InCharge::session; $session =
InCharge::session->init();
$vendor =
    eval{$session ->object( "Router::gw1" ) ->get(Vendor);
};
if ( $@ ) {
    print "Error obtaining the Vendor property\n";
} else {
    print "Vendor is $vendor\n";
}
```

Refer to the section on the *eval* and *die* functions in the Perlfunc manual page for more details about using this mechanism.

All error messages thrown by the API start with a number in square brackets. This is the error code and classifies the error as being one of those listed in the table below. The remainder of the error text gives a verbose description of the specific error that was thrown. Where additional numeric codes are relevant, these are included in a second or subsequent set of square brackets.

An example of a script is provided that attempts a connection with a CNCC Manager and prompts for a username and password if the connection fails due to an authentication error: code 4.

```
my $domain = "SAM1";
my $user = undef;
my $passwd = undef;
for ( ; ; ) {
    $session = eval{ InCharge::session->new(
        domain => $domain,
        username => $user,
        password => $passwd);
    }
    if ( $@ =~ m/^\[4\]/ ) {
        print "Login: ";    chomp $user    = <STDIN>;
        print "Password: "; chomp $passwd = <STDIN>;
    } elsif ( $@ ) {
```



```

        die $@; # Some other fault
    } else {
        last;   # Success !
    }
}

```

## Error codes

ERROR CODE	ERROR TYPE	DESCRIPTION
1	Syntax error	Wrong number of arguments, missing argument, or too many arguments
2	System error	System call error; e.g., socket creation failed
3	Connection error	Socket connection error
4	Authentication error	Authentication error
5	HTTP error	Other session init failure (HTTP error in second number; e.g., “[5][301]”)
6	Bad argument	Argument content or type error, or invalid name, invalid option, or wrong type; e.g., is scalar but reference wanted
7	Broker error	Cannot attach to broker
8	No domain	Domain not registered with broker
9	Protocol error	Protocol error, data size error, or unsupported protocol format
10	Isolated	Not attached
11	Invalid operation	Invalid or illegal operation
12	Bad function	Bad function call or primitive name
13	IO error	Socket IO error
14	Timeout	Timer expired
15	DM error	Error returned by CNCC Manager
16	Not cached	Reply missing from cache
17	Configuration error	A required configuration element, such as an environment variable, is either missing or contains invalid data

Table 20: **Error Codes**

## Data Types

The names of the variables used in the primitive descriptions below to denote the arguments and return values indicate the data type passed or expected. Although every effort has been made to use self-descriptive argument names in this document, one or two of them need further explanation.

### **\$session**

This is a reference to a valid *InCharge::session* object - created using *InCharge::session->new()* or *InCharge::session->init()*. All CNCC primitives should be called with reference to an *InCharge::session* object, as shown:

```
$session = InCharge::session->init();  
@list = $session->getClassInstances( "Router" );
```

### **\$object**

The specification of an repository object to be acted upon.

This can be given in one of the following formats:

#### **'class::instance'**

This format uses a single string, containing both the class and instance name with two colons between them.

```
$n =  
    $session->countElements( "Router::gw1", "ComposedOf" );
```

#### **'::instance'**

This format uses a single string, as above, but with the empty class name. In this case the API performs a database query to determine the name of the class to use. This syntax can only be used to refer to existing objects.

```
$n = $session->countElements( "::gw1", "ComposedOf" );
```

#### **\$class, \$instance**

This format uses two parameters, where the first contains the class and the second contains the name of the instance.

```
$n =  
    $session->countElements( "Router",  
                            "gw1", "ComposedOf" );
```

**undef, \$instance**

This format uses two parameters, as above, but with the first replaced by the Perl *undef* value, indicating that it is unknown. This causes the API to perform a query to determine that name of the object's class. This syntax can only be used to refer to existing objects.

```
$n = $session->countElements(undef, "gw1", "ComposedOf");
```

**InCharge::object reference**

This format is used whenever an object name is required. It is also possible to pass an *InCharge::object* reference.

```
$obj = $session->object( "Router::gw1" );  
$n = $session->countElements( $obj, "ComposedOf" );
```

## @objects

A list of objects is to be returned, which is only used as an return type.

The return is an array of object name strings in the format  
“ClassName::InstanceName”.

## \$symptom, @symptoms

A number of calls return lists of symptoms. These are represented as an array of array references. Each sub-array consists of four elements, each of which has the following significance:

```
$x[0] = type (INT)  
$x[1] = certainly (FLOAT)  
$x[2] = object (STRING - class::instance)  
$x[3] = event/symptom name (STRING)
```

You can gain access to the elements using the syntax,

```
$list[ $record_number ] -> [ $field_number ]
```

or,

```
$listref -> [ $record_number ] -> [ $field_number ]
```

The first of these is for use where the list is held in an array variable. The second is used when the list is held in an array pointed to by a reference.

# **\$symptomData, @symptomData**

Symptom data is returned as an array of nine values, as described below. When a list of symptoms is returned, it is formatted as an array of array references where each sub-array contains the nine fields for a single symptom.

SYMPTOM DATA CODE	DESCRIPTION AND TYPE
\$x[0]	state (INTEGER) 0 = active 1 = inactive 2 = suspended 3 = not monitored
\$x[1]	last occurrence (LONG INTEGER)
\$x[2]	instance display name (STRING)
\$x[3]	class display name (STRING)
\$x[4]	event type (INTEGER)
\$x[5]	event certainty level (FLOAT)
\$x[6]	event class (STRING)
\$x[7]	event instance (STRING)
\$x[8]	event name (STRING)

Table 21: **Symptom Data Codes**

# **\$type, @types**

The CNCC Manager protocol uses a range of integer values to identify the types of data being passed. These are used when a primitive is permitted to handle more than one data type as an argument or return value. For example, the *invoke()* primitive can take arguments of any type, such as integer, string, and boolean. When specifying a type as a primitive function argument you can either use the numeric value or the mnemonic string, shown below. For a string, either use “13” or “STRING”. When type codes are returned by primitives, they are always returned as the numeric code.

To convert from the numeric code to the mnemonic string and back, use the built-in TYPE method of the *InCharge::session* module, as shown.

```
$mnemonic = $session->TYPE( $code )
```

or,

```
$code = $session->TYPE( $mnemonic );
```

The type code values used are:

CONSTANT	LITERAL	DESCRIPTION
0	VOID	void (nothing)
1	ERR	error condition
2	BOOLEAN	boolean (1 = true, 0 = false)
3	INT	signed integer
4	UNSIGNED	unsigned integer
5	LONG	signed long integer
6	UNSIGNEDLONG	unsigned long integer
7	SHORT	signed short integer
8	UNSIGNEDSHORT	unsigned short integer
9	FLOAT	floating point
10	DOUBLE	double length floating point
12	CHAR	1-byte character
13	STRING	string
14	OBJREF	object (class and instance)
15	OBJCONSTREF	constant object reference
16	BOOLEAN_SET	set of booleans
17	INT_SET	set of signed integers
18	UNSIGNED_SET	set of unsigned integers
19	LONG_SET	set of signed long integers
20	UNSIGNEDLONG_SET	set of unsigned long integers
21	SHORT_SET	set of signed short integers
22	UNSIGNEDSHORT_SET	set of unsigned short integers
23	FLOAT_SET	set of floating point numbers
24	DOUBLE_SET	set of double length floats
26	CHAR_SET	set of 1-byte characters
27	STRING_SET	set of strings

CONSTANT	LITERAL	DESCRIPTION
28	OBJREF_SET	set of objects (class and instance)
29	OBJCONSTREF_SET	set of constant object references
30	ANYVALARRAY	set of values (types included)
31	ANYVALARRAY_SET	two-dimensional array of values

Table 22: **Type Codes**

## \$freshness

Where the function argument list takes a freshness parameter, this refers to how fresh the property being accessed by the function should be. This applies to polled or derived properties that may need re-calculating or re-polling if the property was last updated more than the specified \$freshness seconds ago.

# Primitives

## classExists

```
$boolean = $session->classExists( $class )
```

Returns 1 if the specified class exists or 0, otherwise.

```
if ( $session->classExists( "Router" ) ) {  
    print "Router class exists\n";  
}
```

## consistencyUpdate

```
$session->consistencyUpdate( )
```

The *consistencyUpdate()* function causes the CNCC Manager to recompute the correlation rules.

## correlate

```
$session->correlate( )
```

Triggers the CNCC ``Code book'' correlation actions, where symptoms are analyzed and correlated into problems.

## countChildren

```
$count = $session->countChildren( $class )
```

Counts the child classes of the specified class.

```
$class = "ICIM_UnitaryComputerSystem";  
$n = $session->countChildren($class);
```

## countClassInstances

```
$count = $session->countClassInstances( $class )
```

Counts the number of objects that exist for a specified class, or those that would be returned by a call to *getClassInstances()*.

```
$n = $session->countClassInstances( "Router" );
```

## countClasses

```
$count = $session->countClasses( )
```

Counts the number of classes present in the system.

```
$n = $session->countClasses( );
```

## countElements

```
$count = $session->countElements( $object, $relation )
```

Counts the number of elements in the specified relationship.

```
$n = $session->countElements("Router::gw1", "ComposedOf");
```

## countInstances

```
$count = $session->countInstances( )
```

Counts the total number of objects in the repository, of all classes.

```
$n = $session->countInstances( );
```

## countLeafInstances

```
$count = $session->countLeafInstances( $class )
```

Counts the number of leaf objects that exist for a specified class, those that would be returned by a call to *getLeafInstances()*.

```
$n = $session->countLeafInstances( "Router" );
```

## countf

```
$count =  
    $session->countf( $object, $relationship, $freshness )
```

Counts the number of elements in the specified relationship, such as *countElements()*. The contents of the relationship will be refreshed if the values are older than \$freshness seconds.

## createInstance

```
$session->createInstance( $object )
```

---

**Note:** Use the *InCharge::session->create()* function instead, as described in the Session chapter of this guide.

---

Creates a new ICIM object instance. The object specification must include both a class name and unique instance name.

```
$session->createInstance( "Router::fred" );
```

## deleteInstance

```
$session->deleteInstance( $object )
```

---

**Note:** Use the *InCharge::session->delete()* or *InCharge::object->delete()* function instead, as described in the Session or Object chapters of this guide.

---

Deletes the specified ICIM object instance from the database. Note that this does not clean up all the object inter-dependencies and links. For a cleaner object deletion, use the remove operation, if one exists, for the object class in question; see *invoke()*.

```
$session->deleteInstance( "ACT_File::myFile" );
```

## deleteObserver

```
$session->deleteObserver( )
```

Alias for *purgeObserver()*.

---

**Note:** Consider this an internal call - use *\$session->detach()* instead, as discussed in the Session chapter of this guide.

---



Reverses the effect of *getObserverId()*, deregistering the script as an observer.

```
$session->deleteObserver();
```

## eventIsExported

Alias for *getEventExported()*.

## execute

Alias for *executeProgram()*.

## executeProgram

```
@thread = $session->executeProgram( $program, \@args )
```

Alias for *execute()*

Executes a CNCC program, passing arguments to it.

The following example runs the dmdebug plug-in, displaying statistics information on the stdout file of the sm\_server process.

```
@thread = $session->executeProgram (
    "dmdebug", [ "dmdebug", "--stats" ] );
```

## exists

Alias for *instanceExists()*.

## findInstances\_P

---

**Note:** Use the *findInstances()* function from the *InCharge::session* module instead.

---

```
@objects = $session->findInstances_P(
    $class-pattern, $instance-pattern, $flag )
```

Finds instances that match the class and instance patterns, according to rules specified in the flags.

When used by the console GUI, the \$flag value is 0x101000, which requests subclass expansion and glob pattern matches. When used by dmctl, the value 0x001000 is used which requests RegEx pattern matches and sub class expansion.

The value of \$flag consists of the following values or'd in any combination, according to the options required.

0x001000 = Expand-subclasses. With this flag set, the contents of subclasses of classes that match are also returned.

0x100000 = Glob. This causes the match to be done using ICIM *glob( )* matches rather than unix regex syntax, which is used otherwise.

```
@list = $session->findInstances_P(  
                                "Router", "s*", 0x100000 );  
@list = $session->findInstances_P(  
    "ICIM_UnitaryComputerSystem", ".*", 0x001000);
```

## forceNotify

`forceNotify( $object, $event, $notified, $expires)`

Notifies, or clears, the specified event.

The `$notified` and `$expires` parameters are both timers. If `$notified` is greater-than-or-equal-to `$expires`, then the event is cleared. If `$notified` is less-than `$expires` then the event is notified, or raised. The actual values of these parameters are not significant.

```
# to notify an event:  
$session->forceNotify("Router::gw1",  
                     "Unresponsive", 0, 1);  
# to clear an event:  
$session->forceNotify("Router::gw1",  
                     "Unresponsive", 0, 0);
```

## get

`RETURN = $session->get( $object, $property )`

---

**Note:** Use the features of the *InCharge::object* module instead, as discussed in the Object and Session chapters of this guide.

---

Gets the contents of the specified property of the object.

The return type is scalar, array, or array reference as appropriate.

```
$vendor = $session->get( "Router::gw1", "Vendor" );  
@parts = $session->get( "Router::gw1", "ComposedOf" );
```

The preferred implementation is:

```
$object = $session->object( "Router::gw1" );  
$vendor = $object->{Vendor};  
@parts = $object->{ComposedOf};
```

## get\_t and get\_T

```
($type, $value) = $session->get_t( $object, $property )
```

Like *get()*, above, this returns the contents of the specified property, however, *get\_t()* also returns a code for the type of the data. The returned value will be a scalar or array reference, as appropriate.

```
( $type, $value ) = $session->get_t(
    "Router::gw1", "Vendor" );
( $type, $value ) = $session->get_t(
    "Router::gw1", "ComposedOf" );
```

The *get\_t()* variant of this call also returns the types of values contained in complex structures. Where *get\_t()* would return a value, *get\_T()* returns a type code and value in a two-element array.

## getAggregationEvents

```
@list = $session->getAggregationEvents( $object,
                                         $eventname, $flag)
```

Gets the names of the events that are aggregated to the specified event, which must be an aggregation event type.

If \$flag is false then the events directly aggregated are returned. If \$flag is true, then the aggregation tree is walked, and the names of all non aggregation events that the specified event ultimately depends on are returned.

```
@list = $session->getAggregationEvents(
    "Router::gw1", "PowerSupplyException", 1 );
```

## getAllEventNames

```
@events = $session->getAllEventNames( $class )
```

Alias for *getEvents()*.

Gets the list of all events of all types, including symptoms, problems, aggregates, and events, in no particular order.

The *getEventNames()* call is similar but omits the problems from the list.

```
@list = $session->getAllEventNames( "Router" );
```

## getAllInstances

```
@instances = $session->getAllInstances( )
```

Gets the names of all instances present in the ICIM database.

---

**Note:** This potentially returns a very large array indeed and should, therefore, be avoided.

---

## getAllProperties and getAllProperties\_t

```
@properties = $session->getAllProperties( $object, $flag );
```

Returns the names and values of all the properties of the specified object.

If \$flag is 0, attributes only are returned. If \$flag is 1, relations only are returned. If \$flag is 2, both attributes and relations are returned.

The @properties array contains an even number of elements, where the odd numbered ones are the property names and the even numbered are the matching values. This convention means that you can treat the result as a Perl hash, as shown in the example.

```
%props = $s->getAllProperties( $obj, 2 );  
print "Object Name is $props{Name}\n";
```

or,

```
use Data::Dumper;  
print Dumper( \%props );
```

The “\_t” variation of the call returns data types as well as values.

Consider using the *get()* or *get\_t()* functions of the *InCharge::object* module with no arguments instead of this call, as shown.

```
%props = $obj->get( );  
print Dumper( \%props );
```

## getArgDirection

```
$direction = $session->getArgDirection( $class, $operation,  
$argname )
```

Gets a flag to indicate whether the specified operation argument is an IN or OUT argument.

IN arguments are denoted by the value 0 and refer to argument values passed from the script to the CNCC Manager. OUT arguments are denoted by the value 1 and refer to variables into which the operation puts result information.

Nearly all argument to all operations of all classes are IN arguments.

---

**Note:** OUT arguments are not supported by the remote access protocol; this discussion is beyond the scope of the API, dmctl, and ASL.

---

```
$direction = $session->getArgDirection(  
    "Router", "getFan", "identifier" );
```

## getArgType

```
$type = $session->getArgType( $class, $operation, $argname )
```

Alias for *getOpArgType()* and *getOperationArgumentType()*.

Gets the type of the specified argument for the specified class operation.

Refer to the discussion about data types for a description of the possible values.

```
$type =  
    $session->getArgType( "Router", "makeFan", "className" );
```

## getAttributes

Alias for *getAttributeNames()*.

## getAttributeNames

```
@properties = $session->getAttributeNames( $class )
```

Alias for *getAttributes()*.

Gets the list of all attributes for the specified class.

Attributes are properties that are not relations. For class Router, Vendor is an attribute but ComposedOf is not; however, both are properties. The *getAttributeTypes()* call returns the types of these attributes.

```
@list = $session->getAttributeNames( "Router" );
```

## getAttributeTypes

```
@types = $session->getAttributeTypes( $class )
```

Gets the list of type codes associated with the attribute names returned by *getAttributeNames()*.

The types returned by this call and the names returned by *getAttributeNames()* are in the same order, such that the type of *\$property[\$n]* is given in *\$type[\$n]*. Refer to the discussion about data types for a description of the possible values.

```
@list = $session->getAttributeTypes( "Router" );
```

## getByKey

```
RESULT = $session->getByKey( $object, $table,  
                             [ $keytype, $keyvalue ] )
```

Gets the entry in the named table from the object, indexed by its key.

Tables are properties that can contain arrays of values.

```
@driver = $session->getByKey(  
    "GA_CompoundDriver::Bridge-Generic-Driver",  
    "drivers", [ "INT", 10 ] );
```

## getByKey\_t and getByKey\_T

```
($type, $value) = $session->getByKey_t( $object, $table,  
                                         [ $keytype, $keyvalue ] )
```

Identical to *getByKey()* but returns a code for the type of the result as well.

```
($type, $data) = $session->getByKey_t(  
    "GA_CompoundDriver::Bridge-Generic-Driver",  
    "drivers", [ "INT", 10 ] );
```

## getByKeyf

```
RESULT = $session->getByKeyf( $object, $table,  
                              [ $keytype, $keyvalue ], $freshness )
```

Identical to *getByKey()* but takes the “freshness” of the entry into account.

```
@driver = $session->getByKeyf(  
    "GA_CompoundDriver::Bridge-Generic-Driver",  
    "drivers", [ "INT", 10 ], 120 );
```

## getByKeyf\_t and getByKeyf\_T

```
($type, $value) = $session->getByKeyf_t( $object, $table,  
                                          [ $keytype, $keyvalue ], $freshness )
```

Identical to *getByKey\_t()* but takes the “freshness” of the entry into account.

```
($type, $data) = $session->getByKeyf_t(  
    "GA_CompoundDriver::Bridge-Generic-Driver",  
    "drivers", [ "INT", 10 ], 120 );
```

## getChildren

```
@classes = $session->getChildren( $class )
```

Gets the list of classes that are child classes of a specified one; i.e., classes derived from the base class.

```
$class = "ICIM_UnitaryComputerSystem";  
@list = $session->getChildren($class);
```

## getClassDescription

```
$text = $session->getClassDescription( $class )
```

Gets a textual description of the class.

The fixed string “no description available” is returned if the class programmer has not provided a description message for the class.

```
$description = $session->getClassDescription( "Router" );
```

## getClassHierarchy

```
@hierarchy = $session->getClassHierarchy( );
```

Returns an array of information that provides a complete description of the hierarchy of classes.

Each element of the array is a reference to a three-element subarray, the fields of which have the following significance.

ARRAY ELEMENT	DESCRIPTION
<code>\$x[0]</code>	name of ICIM class
<code>\$x[1]</code>	name of the class’s parent class
<code>\$x[2]</code>	class is abstract flag: 1 = yes, 0 = no

Table 23: **Class Hierarchy Descriptor**

## getClassInstances

```
@instances = $session->getClassInstances( $class )
```

Gets the list of instances of a specified class.

The return is a list of strings which contain the instance names without the class name. For example, “fred” is returned rather than “Router::fred”. This differs from *getLeafInstances()* in that this call returns the members of the class and any derived classes, whereas *getLeafInstances()* returns only the members of the specified class.

```
@names = $session->getClassInstances( "Router" );
```

## getClasses

```
@classes = $session->getClasses( )
```

Gets the list of class present in the system.

The following code fragment displays the list of all instances of all classes in the database.

```
foreach $class ($session->getClasses()) {  
    foreach ($session->getClassInstances($class)) {  
        print "${class}::$_\n";  
    }  
}
```

## getCorrelationParameters

```
@info = $session->getCorrelationParameters( )
```

Returns a nine-element array, each element of which contains a parameter relating to the CNCC Manager correlation mechanism.

The array elements have the following significance:

ELEMENT	DESCRIPTION
info[0]	max problems (INT)
info[1]	correlation interval (INT)
info[2]	codebook radius (FLOAT)
info[3]	correlation radius (FLOAT)
info[4]	lost symptom probability (FLOAT)



ELEMENT	DESCRIPTION
info[5]	spurious symptom probability (FLOAT)
info[6]	time limit (INT)
info[7]	suspend correlation (BOOLEAN)
info[8]	provide explanation (BOOLEAN)

Table 24: **getCorrelationParameters Return Values**

## getEnumVals

```
@strings = $session->getEnumVals( $class, $property )
```

Returns the list of strings representing the possible values for an enumerated property.

This can be used to present a list of valid values to the user in the form of a selection menu. If this primitive is used to refer to an property that is not an enumerated one, and error is thrown.

```
@values = $session->getEnumVals( "Router", "Type" );
```

## getEvents

Alias for *getAllEventNames()*

## getEventCauses

```
@symptoms =
    $session->getEventCauses( $object, $eventname, $flag )
```

Gets a list of the Root causes, or problems, that the specified event can be considered to be a symptom of.

The *getProblemClosure()* primitive provides the reverse mapping. This is the mechanism used to populate the codebook tab for an event property sheet in the administrative console.

The \$flag parameter is optional. If it is passed as TRUE, the full list of problems explaining eventname, whether directly or indirectly, is returned. If it is passed as FALSE, only those problems that directly list eventname among the events they explain are returned.

```
@causes = $session->getEventCauses(
    "Router::gw1", "MightBeUnavailable", 1 );
```

## getEventClassName

```
$class = $session->getEventClassName( $class, $event )
```

The *getEventClassName()* function returns a string with the name of the ancestor class associated with a class and an event. The ancestor class is where the event was originally defined; i.e., the class in which the event definition statement, not any refinement, appeared.

```
$class = $session->getEventClassname( "Router", "Down" );
```

## getEventDescription

```
$text = $session->getEventDescription( $class, $event )
```

The *getEventDescription()* function returns a string, defined in MODEL, that describes an event.

```
$descr = $session->getEventDescription("Router", "Down");
```

## getEventExplainedBy

```
@symptoms =  
  $session->getEventExplainedBy( $object, $event, $flag )
```

---

**Note:** Use *InCharge::session->getExplainedBy* instead, as discussed in the Session chapter of this guide.

---

Returns the list of symptoms that are explained by the specified impact event.

The \$flag is a boolean that indicates whether the event impact tree is to be walked during the processing of the request.

```
@list = $session->getEventExplainedBy(  
    "Router::gw1", "DownImpact", 1);
```

## getEventExported

```
$boolean = $session->getEventExported( $class, $event )
```

Alias for *eventIsExported()*.

Returns 1 if the specified event is exported by the class or 0 if it is not exported. Unexported events are hidden from view in the GUI.

```
if ( $session->getEventExported( "Router", "Down" ) ) {  
    print "Event is exported\n";  
}
```

## getEventNames

```
@events = $session->getEventNames( $class )
```

Gets the list of events handled by the specified class.

Some of the returned events are exported while others are not as described in *getEventExported()*. Unlike *getAllEventNames()*, this call does not return problems names.

```
@list = $session->getEventNames( $class );
```

## getEventSymptoms

```
@events = $session->getEventSymptoms( $class, $event )
```

Returns the list of events that are symptoms of the specified one.

```
@symptoms =  
    $session->getEventSymptoms( "Router", "Down" );
```

## getEventType\_P

```
$eventtype = $session->getEventType_P( $class, $event )
```

---

**Note:** Consider using *InCharge::session->getEventType* instead, as discussed in the Session chapter of this guide. The “\_P” is included in the name in order to avoid confusion with the *InCharge::session* function of the same name.

---

This primitive returns a numeric code that indicates the type of the specified event. Possible values are shown in the table.

RETURN CODE	EVENT TYPE
0	Event
1	Aggregation
2	Symptom
3	Causality
4	Problem
5	Imported event
6	Propagated Aggregation
7	Propagated Symptom
8	Same type

Table 25: **getEventType** Return Codes

```
$seventtype = $session->getEventType_P( "Router", "Down");
```

---

**Note:** The primitive version of this call must be called using the primitive function as shown because there is an *InCharge::session* method of the same name, provided for functional compatibility with ASL.

---

## getInstances

Alias for *getClassInstances()*.

## getInstrumentationType

```
$type = $session->getInstrumentationType( $object )
```

Returns the instrumentation type for a specified object.

```
$type =  
    $session->getInstrumentationType( "Router::gw1" );
```

## getLeafInstances

```
@instances = $session->getLeafInstances( $class )
```

Alias for *getInstances()*.

Gets the list of instances of a specified class.

The return is a list of strings which contain the instance names without the class name. For example, “fred” is returned rather than “Router::fred”. This differs from *getClassInstances()* in that this call returns only the members of the specified class, whereas the *getClassInstances()* call returns the members of the class and its derived classes.

```
@names = $session->getLeafInstances( "Router" );
```

## getLibraries

```
@libs = $session->getLibraries( )
```

Alias for *getModels()*.

Gets the list of libraries loaded into the system.

## getModels

Alias for *getLibraries()*

## getMultipleProperties and getMultipleProperties\_t

---

**Note:** Use the *get()* and *get\_t()* functions of the *InCharge::object* module with multiple arguments instead of this call, as shown in the example.

---

```
( $vendor, $model ) = $obj->get( "Vendor", "Model" );
```

The syntax of the primitive itself is,

```
@values =  
$session->getMultipleProperties( $object, \@propnames);
```

or,

```
@values =  
$session->getMultipleProperties_t($object, \@propnames);
```

For example,

```
( $vendor, $model ) =  
$session->getMultipleProperties( $obj,  
                                [ "Vendor", "Model" ] );
```

The argument is a reference to an array containing the names of the properties to be returned.

## getObserverId

```
$id = $session->getObserverId()
```

---

**Note:** Do not call this primitive directly, but make use of the *InCharge::session->observer()* function instead.

---

Creates and returns a new observer id.

The *deleteObserver()* primitive reverses this action.

## getOpArgType

Alias for *getArgType()*.

## getOpArgs

```
@argnames = $session->getOpArgs( $class, $operation )
```

Alias for *getOperationArguments()*.

Gets the names of the arguments for a specified class operation.

The argument names are returned in the order in which they should appear in the argument list when invoking the operation.

```
@list = $session->getOpArgs( "Router", "makeIP" );
```

## **getOpDescription**

```
$text = $session->getOpDescription( $class, $operation )
```

Alias for *getOperationDescription()*.

Returns a textual description of the specified class operation.

```
$description = $session->getOpDescription(  
    "Router", "makeIP");
```

## **getOperationArguments**

Alias for *getOpArgs()*.

## **getOperationArgumentType**

Alias for *getArgType()*.

## **getOperationDescription**

Alias for *getOpDescription()*.

## **getOperationFlag**

Alias for *getOpFlag()*.

## **getOperationReturnType**

Alias for *getOpReturnType()*.

## **getOperations**

Alias for *getOpNames()*.

## getOpFlag

```
$flag = $session->getOpFlag( $class, $operation )
```

Alias for *getOperationFlag()*.

Gets the flag associated with the specified class operation.

The value returned is between 0 and 3, as defined in the table.

RETURN CODE	OPERATION
0	No flag
1	Idempotent
2	Constant
3	Readonly

Table 26: **getOpFlag Return Codes**

```
$flag = $session->getOpFlag( "Router", "makeIP" );
```

## getOpNames

```
@operations = $session->getOpNames( $class )
```

Alias for *getOperations()*.

Gets the list of operations for the specified class.

The operations are returned as an array of strings containing their names.

```
@list = $session->getOpNames( "Router" );
```

## getOpReturnType

```
$type = $session->getOpReturnType( $class, $operation )
```

Alias for *getOperationReturnType()*.

Returns the return type code for the specified class operation.

Using this function you can determine whether the operation returns an integer, a string, an object, or a list. The type codes returned are integer numbers, as described for \$type in notes about data types, above.

```
$type_code = $session->getOpReturnType(
    "Router", "makeIP" );
```

## getParentClass

```
$class = $session->getParentClass( $class )
```

Returns the name of the class from which the specified class is derived.

This is the logical inverse of *getChildren()*.

```
$parent = $session->getParentClass( "Router" );
```

## getProblemClosure

```
@symptoms =  
    $session->getProblemClosure( $object, $eventname, $flag )
```

---

**Note:** Use *InCharge::session->getClosure()* in preference to this call.

---

Lists the events, or symptoms, that contribute to a specified problem. The *getEventCauses()* primitive is the inverse of this one.

```
@list = $session->getProblemClosure(  
    "Router::gw1", "Down", 1 );
```

## getProblemExplanation

```
@list = $session->getProblemExplanation(  
    $object, $eventname, $flag )
```

---

**Note:** Use *InCharge::session->getExplains()* in preference to this call.

---

MODEL developers can add information to a problem in order to emphasize events that occur because of a problem. This function returns a list of these events.

```
@list = $session->getProblemExplanation( "Router::gw1",  
    "Down", 1 );
```

## getProblemNames

```
@list = $session->getProblemNames( $class )
```

Get the event names of problems associated with the specified class.

```
@problems = $session->getProblemNames( "Router" );
```

## getProblemSymptomState

```
@symptomData = $session->getProblemSymptonState( $object,  
    $eventname )
```



Returns data about all the symptoms that indicate the specified problem, including significant state information.

```
@list =  
    $session->getProblemSymptonState( "Router::gw1", "Down");
```

## getPrograms

```
@list = $session->getPrograms( )
```

Gets the list of ``programs" running in the CNCC Manager.

Typically the reply list includes “dmboot” and “icf.”

```
@progs = $session->getPrograms( );
```

## getPropAccess

```
$access = $session->getPropAccess( $class, $property )
```

Returns a number that indicates the level of access to the specified property.

This effectively identifies the method by which the property value is obtained internally. Possible values and their meanings are:

RETURN CODE	PROPERTY ACCESS LEVEL
0	No access
1	Stored
2	Computed
3	Instrumented
4	Propagated
5	Uncomputable
6	Computed with expression

Table 27: **getPropAccess Return Codes**

```
$access = $session->getPropAccess( "Router", "Vendor" );
```

## getPropDescription

```
$text = $session->getPropDescription( $class, $property )
```

Alias for *getPropertyDescription()*.

Returns a textual description of the named class property.

```
$descr =
```

```
$session->getPropDescription( "Router", "Vendor" );
```

## getProperties

Alias for *getPropNames()*.

---

**Note:** The functionality of the C++ function *getProperties()* is available through the *getMultipleProperties()* primitive, and more easily through the *get()* method of the *InCharge::object* module.

---

Primitive *getProperties()* is aliased to *getPropNames()* in order to provide dmctl syntax compatibility.

## getPropertyDescription

Alias for *getPropDescription()*.

## getProperties

Alias for *getPropNames()*.

---

**Note:** For C++ developers, the C++ API call *getProperties()* is called *getMultipleProperties()*. However, the *InCharge::object->get( )* is an easier way to use this functionality.

---

## getPropertyType

Alias for *getPropType()*.

## getPropIsReadOnly

```
$boolean = $session->getPropIsReadOnly( $class, $property )
```

Indicates whether or not the specified class property is read-only.

```
if ( $session->getPropIsReadOnly( "Router", "Vendor" ) ) {  
    print "Vendor is readonly\n";  
} else {  
    print "Vendor can be changed\n";  
}
```

## getPropIsRelationship

```
$boolean =  
    $session->getPropIsRelationship( $class, $property )
```

Indicates whether or not the specified class property is a relationship.

```
if ( $session->getPropIsRelationship( "Router",  
                                     "ComposedOf" ))  
{  
    print "ComposedOf is a relationship\n";  
}
```

## getPropIsRequired

```
$boolean = $session->getPropIsRequired( $class, $property )
```

Indicates whether or not the specified class property is required to have a value.

```
$needed =  
    $session->getPropIsRequired( "Router", "Vendor" );
```

## getPropNames

```
@list = $session->getPropNames( $class )
```

## getPropRange

```
@range = $session->getPropRange( $class, $property )
```

Returns the range of valid values for the class property, provided the property has been defined.

This applies to a very limited number of properties of integer type, typically in polling configuration classes.

```
( $min, $max ) = $session->getPropRange(  
    "DialOnDemand_Interface_Setting",  
    "MaximumUptime" );
```

## getPropType

```
$type = $seesion->getPropType( $class, $property )
```

Alias for *getPropertyType()*.

Returns the data type for a specified class property.

See the `$type` section of the discussion about data types for a description of the possible values. This call always returns the integer number representation of the type.

```
$type = $session->getPropType( "Router", "Vendor" );
```

## getPropertySubscriptionState

```
$state = $session->getPropertySubscriptionState(  
                                                $object, $property )
```

Gets the current state of subscription to the specified event.

The possible reply values are:

RETURN CODE	SUBSCRIPTION STATE
0	Unsubscribed
1	Pending
2	Subscribed
3	Suspended

Table 28: **getPropertySubscriptionState** Return Codes

## getRelatedClass

```
$class = $session->getRelatedClass( $class, $property )
```

Returns the name of the class of object that can be related to the specified class through the property, which must be a relationship.

```
$class =  
    $session->getRelatedClass( "Router", "ComposedOf" );
```

## getRelationNames

```
@properties = $session->getRelationNames( $class )
```

Alias for *getRelations()*.

Gets the names of all the relationship properties for the specified class.

```
@relationships = $session->getRelationNames( "Router" );
```

## getRelations

Alias for *getRelationNames()*.

## getRelationTypes

```
@types = $session->getRelationTypes( $class )
```

Returns a list of type numbers for the relationships which are returned by the *getRelationNames()* call.

```
@types = $session->getRelationTypes( "Router" );
```

Refer to the notes about data types section for a description of the possible values.

## getReverseRelation

```
$property = $session->getReverseRelation( $class, $property )
```

Returns the name of the other end of a relationship pair denoted by the specified property name.

The inverse of ComposedOf is PartOf.

```
$relationship = $session->getReverseRelation(  
    "Router", "ComposedOf");
```

## getSubscriptionState

```
$state = $session->getSubscriptionState( $object, $event )
```

Gets the current state of subscription to the specified event.

The possible values are:

RETURN CODE	SUBSCRIPTION STATE
0	Unsubscribed
1	Pending
2	Subscribed
3	Suspended

Table 29: **getSubscriptionState** Return Codes

## getThreads

```
@list = $session->getThreads( )
```

Returns a list of threads running in the current CNCC Manager system.

Each element of the returned array is a reference to a four-element array. The four values that describe each thread are, in order:

RETURN ARRAY ELEMENT	THREAD INFORMATION
<code>\$t[0]</code>	Process ID
<code>\$t[1]</code>	name
<code>\$t[2]</code>	State
<code>\$t[3]</code>	Status

Table 30: **getThreads Return Codes**

This example prints the process ID's and names of all threads in process ID order.

```
foreach $t ( sort { $a->[0] <=> $b->[0] }
              $session->getThreads( ) ) {
    print $t->[0] . " - " . $t->[1] . "\n";
}
```

## getf

```
RETURN = $session->getf( $object, $property, $freshness )
```

Gets the contents of the specified property of the object with reference to its freshness; see the discussion of data types. The return type is scalar, array, or array reference, as appropriate.

```
$vendor = $session->getf( "Router::gw1", "Vendor", 240 );
@parts =
    $session->getf( "Router::gw1", "ComposedOf", 360 );
```

## getf\_t and getf\_T

```
($type, $value) =
    $session->getf_t( $object, $property, $freshness )
```

Like *getf()*, this returns the contents of the specified property but *getf\_t()* also returns a code for the type of the data. The returned value will be a scalar or array reference, as appropriate.

```
( $type, $value ) =
    $session->getf_t( "Router::gw1", "Vendor", 240 );
( $type, $value ) =
    $session->getf_t( "Router::gw1", "ComposedOf", 360 );
```

## getfAllProperties and getfAllProperties\_t

```
%properties =  
    $session->getfAllProperties($object, $flag, $freshness);
```

This is the same as *getAllProperties()*, but takes the freshness of the values into account and refreshes any stale properties before returning the results; i.e., those that are older than *\$freshness* seconds.

See *getAllProperties()* for a description of *\$flag*.

## getfMultipleProperties and getfMultipleProperties\_t

```
@values = $session->getfMultipleProperties( $object,  
                                           \@propNames,  
                                           $freshness );
```

Like *getMultipleProperties()*, but refreshes values that are staler than *\$freshness* seconds and need re-polling.

The *propNames* argument must be a reference to an array of property names. For example,

```
@props = qw(Vendor Model Type);  
($v, $m, $t) =  
    $session->getfMultipleProperties($obj, \@props, 30);
```

## hasRequiredProps

```
$boolean = $session->hasRequiredProps( $class )
```

Indicates whether or not the specified class has any properties that are flagged as required.

```
$reqd = $session->hasRequiredProps( "Router" );
```

## insertElement\_P

```
$session->insertElement_P($object,  
                        $relation,  
                        [ $type, $value ])
```

---

**Note:** Use the *InCharge::object::insertElement()* function instead, as described in the Object chapter of this guide. The “\_P” on the end of the primitive name is used to avoid confusion between it and the *InCharge::session* and *InCharge::object* versions of the same call.

---

Inserts something into a relationshipset.

In order to access the low-level primitive version of this call, you must invoke it using the primitive method because the *InCharge::session* module also has its own variant.

```
$session->insertElement_P( "Router", "ComposedOf",  
                           [ "OBJREF", "Fan::fan1" ] );
```

## instanceExists

```
$boolean = $session->instanceExists( $object )
```

Alias for *exists()*.

---

**Note:** Use the ASL-like function *InCharge::object::isNull()* instead of this primitive. Note that the sense of the return value is reversed.

---

Indicates whether or not the named object is present in the repository.

However, the class name and instance name should be specified in the *\$object* parameter.

```
$exists = $session->instanceExists( "Router::gw1" );
```

## invoke

Alias for *invokeOperation()*.

## invoke\_t and invoke\_T

Alias for *invokeOperation\_t()*.

## invokeOperation

```
RESULT =  
    $session->invokeOperation($object, $operation, \@args)
```

Alias for *invoke()*.

---

**Note:** Use the features of the *InCharge::object* module instead, as described in the Object chapter of this guide.

---

Invokes a class operation on a specified object, passing the parameters to the operation.



The syntax of the arguments list requires it to be a reference to an array, each element of which is a reference to a two-element array containing the data type and value. Because of the awkward syntax, using the *InCharge::object* module provides a more natural style of interface.

```
$result = $session->invokeOperation(  
    "Router::gw1", "makeInterface",  
    [  
        [ "INT", 1 ],  
        [ "STRING", "interface-1" ],  
        [ "STRING", "Interface" ]  
    ] );
```

---

**Note:** The argument types, such as INT and STRING in the above example, can be specified using either their type names, as shown above, or numeric codes. See \$type in the DATA TYPES section for the mapping.

---

## invokeOperation\_t and invokeOperation\_T

```
( $type, $value ) =  
    $session->invokeOperation_t($object, $operation, \@args)
```

Alias for *invoke\_t()*.

This is identical to *invokeOperation()* except that the return indicates the type of the returned data as well.

```
( $type, $value ) = $session->invokeOperation_t(  
    "Router::gw1", "makeInterface",  
    [  
        [ "INT", 1 ],  
        [ "STRING", "interface-1" ],  
        [ "STRING", "Interface" ]  
    ] );
```

The “\_T” variation also embeds type codes into the fields of returned complex structures.

## isAbstract

```
$boolean = $session->isAbstract( $class )
```

Indicates whether the specified class is abstract or not.

An abstract class is one from which other classes are derived but which cannot have any objects.

```
$class = "ICIM_UnitaryComputerSystem";  
$flag = $session->isAbstract( $class );
```

## isBaseOf

```
$boolean = $session->isBaseOf( $class1, $class2 )
```

Returns TRUE if \$class2 is a base class of \$class1; i.e., \$class1 is derived from \$class2.

---

**Note:** For the purposes of this query, all classes are taken to be derived from themselves.

---

```
$class1 = "Router";  
$class2 = "ICIM_UnitaryComputerSystem";  
$is_it = $session->isBaseOf( $class1, $class2 );
```

## isBaseOfOrProxy

```
$boolean = $session->isBaseOfOrProxy( $class1, $class2 )
```

Returns TRUE (1) if \$class2 is a base class or proxy class of \$class1.

```
$class1 = "Router";  
$class2 = "ICIM_UnitaryComputerSystem";  
$is_it = $session->isBaseOfOrProxy( $class1, $class2 );
```

## isInstrumented

```
$boolean = $session->isInstrumented( $class )
```

Indicates whether the specified class has associated instrumentation.

```
$flag = $session->isInstrumented( "TCPConnect" );
```

## isMember

```
$boolean = $session->isMember( $object1, $relation, $object2)
```

Returns TRUE if *\$object2* is a member of the specified *\$object1* relationship.

```
$flag = $session->isMember( "Router::strrtbos",  
                           "ComposedOf",  
                           "Interface::IF-strrtbos/1" );
```

## isMemberByKey

```
$boolean = $session->isMemberByKey( $object, $table,  
                                     $keytype, $keyvalue )
```

Indicates whether or not an entry in the named object table exists.

```
$exists = $session->isMemeberByKey(  
    "GA_CompoundDriver::Bridge-Generic-Driver",  
    "drivers", [ "INT", 10 ] )
```

## isMemberByKeyf

```
$boolean = $session->isMemberByKeyf( $object, $table,  
                                     [ $keytype, $keyvalue ], $freshness )
```

Like isMemeberByKey, but with reference to the freshness of the value.

```
$exists = $session->isMemeberByKeyf(  
    "GA_CompoundDriver::Bridge-Generic-Driver",  
    "drivers", [ "INT", 10 ], 120 )
```

## isMemberf

```
$boolean = $session->isMemberf( $object1, $relation,  
                                $object2, $freshness )
```

Returns TRUE if *\$object2* is a member of the specified *\$object1* relationship. If the *\$relation* is a computed or polled value and is more than *\$freshness* seconds old, it is refreshed first.

```
$flag = $session->isMemberf( "Router::strrtbos",  
                             "ComposedOf",  
                             "Interface::IF-strrtbos/1",  
                             240 );
```

## isSubscribed

```
$boolean = $session->isSubscribed( $object, $event )
```

Returns TRUE if the specified event has been subscribed to by the calling process.

```
$subscribed = $session->isSubscribed( "Router::gw1", "Down");
```

## loadLibrary

```
$session->loadLibrary( $library )
```

Alias for *loadModel()*.

Loads a library, model, into sm\_server memory.

```
$session->loadLibrary( $libname );
```

## loadModel

Alias for *loadLibrary()*

## loadProgram

```
$session->loadProgram( $program )
```

Loads the named program into sm\_server memory.

```
$session->loadProgram( "dmdebug" );
```

## noop

```
$session->noop()
```

Alias for *ping()*

This is a type of ping. It sends a null command string to the CNCC Manager, thus determining whether the client/server link is active.

## notify

Information to follow.

## ping

Alias for *noop()*

## propertySubscribe

```
$session->propertySubscribe( $object, $property, $interval )
```

Subscribes to notifications of changes to the specified object property.

See the introductory discussion of the subscription. The actions of this call are reversed by *propertyUnsubscribe()*.

```
$session->propertySubscribe( "Router::gw1",  
                             "Vendor", 30 );
```

## propertySubscribeAll

```
$session->propertySubscribeAll( $flags,$class_pattern,  
                               $instance_pattern,  
                               $property_pattern, $interval);
```

Subscribes to changes in all the matching properties in the matching objects.

The meaning of the \$flags is described for the *subscribe()* session function.

The actions of this call are reversed by *propertyUnsubscribeAll()*.

```
$session->propertySubscribeAll( 0, "Router", "gw1",  
                               ".*", 30 );
```

## propertyUnsubscribe

```
$session->propertyUnsubscribe( $object, $property )
```

Reverses the effect of the *propertySubscribe()* call.

```
$session->propertyUnsubscribe( "Router::gw1", "Vendor" );
```

## propertyUnsubscribeAll

```
$session->propertyUnsubscribeAll( $flags, $class_pattern,  
                                 $instance_pattern,  
                                 $property_pattern );
```

Unsubscribes from changes in all the matching properties in the matching objects.

The meaning of the \$flags is described for the *subscribe()* session function.

## purgeObserver

Alias for *deleteObserver()*.

## put\_P

```
$session->put_P( $object, $property, [ $type, $value ] )
```

Writes the specified value to the specified object property.

This is the low-level primitive that the *put()* function of *InCharge::session* uses and is called when using the hash dereferencing syntax of *InCharge::object*.

The reader is encouraged to use the *InCharge::object* logic.

The following are essentially equivalent.

```
$obj = $session->object( "Router::gw2" );  
$obj->{Vendor} = "Cisco";
```

or,

```
$obj->put( "Vendor", "Cisco" );
```

or,

```
$obj->put( Vendor => "Cisco",  
          PrimaryOwnerContact => "Joe Bloggs" );
```

or,

```
$session->put( "Router::gw", "Vendor", "Cisco" );
```

or,

```
$session->put_P( "Router::gw", "Vendor",  
               [ "STRING", "Cisco" ] );
```

## quit

```
$session->quit( )
```

Alias for *shutdown()*.

Closes down the CNCC Manager cleanly, saving the configured parts of the repository to disk.

## removeElement\_P

```
$session->removeElement_P( $object, $relation,  
                          [ $type, $value ] )
```

---

**Note:** Use *InCharge::object::removeElement()* instead of this primitive. The “\_P” in the name is there to save ambiguity between this primitive and the *InCharge::session* and *InCharge::object* equivalents.

---

This call removes an element from an object relationship, such as *ComposedOf*.

In order to access the low-level primitive version of this call, invoke it using the primitive method because the *InCharge::session* module has a method of the same name that provides an enhanced interface.

```
$session->removeElement_P("Router::gw", "ComposedOf",  
                           [ "OBJREG", "Host::pingu6" ] );
```

## removeElementByKey

```
$session->removeElementByKey( $object, $stable,  
                             [ $keytype, $keyvalue ] )
```

Remove a set-valued property by key.

## restoreRepository

```
$session->restoreRepository( $filename, $purgeflag )
```

Restores the repository from file, optionally purging existing repository contents in the process.

```
$session->restoreRepository( "save.rps", 0 );
```

## setCorrelationParameters

```
$session->setCorrelationParameters( @info )
```

Sets the CNCC Manager correlation parameters.

For a description of the fields of the @info array, refer to the *getCorrelationParameters()* primitive. The example below sets the correlation interval to 20 seconds.

```
@info = $session->getCorrelationParameters( );  
$info[1] = 20;  
$session->setCorrelationParameters( @info );
```

## shutdown

Alias for *quit()*.

## storeAllRepository

```
$session->storeAllRepository( $filename )
```

---

**Note:** Use the *InCharge::session->save()* method instead.

---

Saves the repository in the named file, which is located in the directory `$SM_HOME/repos`. The directory name must not contain any path separator characters.

```
$session->saveAllRepository( "save.rps" );
```

## storeClassRepository

```
$session->storeClassRepository( $filename, $class )
```

---

**Note:** Use the *InCharge::session->save()* method instead.

---

Save the repository for the named class in the specified file.

```
$session->saveClassRepository( "save.rps", "Host" );
```

## subscribeEvent

```
$session->subscribeEvent( $object, $event )
```

Subscribes to a specific event without using wildcard pattern matching, unlike *subscribeAll()*.

The function *unsubscribeEvent()* cancels subscriptions established using *subscribeEvent()*.

```
$session->subscribeEvent( "Router::gateway39", "Down" );
```

## subscribeAll

---

**Note:** Use *InCharge::session->subscribe()* instead.

---

## topologySubscribe

```
$session->topologySubscribe( )
```

Subscribes to notifications of topology updates.

The subscription/observer mechanism is described in detail in the introductory discussion of subscriptions. API subscriptions topology subscriptions may be reversed using *topologyUnsubscribe()*.

## topologyUnsubscribe

```
$session->topologyUnsubscribe( )
```



Cancels topology subscriptions previously requested with the *topologySubscribe()* function.

```
$session->topologyUnsubscribe( );
```

## transactionAbort

```
$session->transactionAbort( )
```

Use *InCharge::session>abortTxn* instead.

Aborts a transactional block previously started using *transactionStart()*.

## transactionCommit

```
$session->transactionCommit( )
```

---

**Note:** Use *InCharge::session->commitTxn()* instead.

---

Commits a transactional block previously started using *transactionStart()*.

## transactionStart

```
$session->transactionStart( $lock_code )
```

---

**Note:** Use *InCharge::session->transaction()* instead.

---

Starts a transaction block, which may subsequently be aborted or committed using *transactionAbort()*/*transactionCommit()*. The *\$lock\_code* values have the following possible values:

LOCK CODE	LITERAL
0	SM_NO_LOCK
1	SM_READ_LOCK_ONLY
2	SM_READ_LOCK
3	SM_WRITE_LOCK

Table 31: **Lock Code Literals**

```
sub SM_READ_LOCK { 1 };
$session->transactionStart( SM_READ_LOCK );
```

## unsubscribeAll

---

**Note:** Use *InCharge::session->unsubscribe()* instead.

---

## unsubscribeEvent

```
$session->unsubscribeEvent( $object, $event )
```

---

**Note:** Use *InCharge::session::unsubscribe()* instead.

---

Unsubscribes from the event previously subscribed using `subscribeEvent()`.

# Index

## Symbols

\$freshness 56  
\$object 52  
\$session 52  
\$symptom 53  
\$type 54  
@objects 53  
@symptoms 53  
@types 54

## A

abortTxn 42, 43, 91  
API xiv, 1, 2, 3, 4, 5, 10, 18, 27, 28, 33, 43, 47, 48, 49, 50, 52, 53, 63, 76, 90  
ASL xiv, 1, 3, 5, 9, 17, 18, 22, 26, 27, 31, 33, 35, 39, 40, 41, 47, 48, 63, 70, 82  
Attribute 1, 2, 62, 63

## B

broken 30

## C

callPrimitive 33, 34, 36, 37  
Certainty 9, 10, 54  
Class 1, 2, 3, 9, 10, 11, 12, 14, 18, 19, 23, 32, 33, 35, 37, 38, 39, 40, 41, 44, 52, 53, 54, 55, 56, 57, 58, 59, 60, 63, 65, 66, 68, 69, 70, 72, 73, 74, 75, 76, 77, 78, 81, 82, 83, 84, 90  
classExists 56  
clear 24  
CNCC  
    Client xiv, 47, 86  
    Manager xiv, 47, 48, 51, 54, 56, 62, 66, 75, 89  
    Server xiv, 1, 79, 86  
commitTxn 42, 43, 44, 91  
consistencyUpdate 56  
correlate 56  
countChildren 57  
countClasses 57  
countClassInstances 57  
countElements 24, 52, 53, 57, 58  
countf 58  
countInstances 57

countLeafInstances 57  
create 18, 25, 33, 58  
createInstance 33, 58

## D

delete 24, 43, 44, 58  
deleteInstance 58  
deleteObserver 58, 59, 71, 87  
detach 4, 5, 7, 30, 49, 58  
die 14, 27, 49, 50, 51  
dmctl 1, 19, 26, 27, 37, 47, 48, 59, 63, 76

## E

eval 27, 49, 50  
eventIsExported 59, 68  
execute 59  
executeProgram 59  
exists 59, 82

## F

findInstances 38, 59  
findInstances\_P 59, 60  
forceNotify 60

## G

get 3, 17, 18, 19, 20, 21, 49, 50, 60, 61, 62, 71, 76  
get\_T 61  
get\_t 21, 61, 62, 71  
getAggregationEvents 61  
getAllEventNames 61, 67, 69  
getAllInstances 61  
getAllProperties 62, 81  
getAllProperties\_t 62  
getArgDirection 62, 63  
getArgType 63, 71, 72  
getAttributeNames 63  
getAttributes 63  
getAttributeTypes 63, 64  
getByKey 64  
getByKey\_t 64  
getByKeyf 64

getByKeyf\_T 64  
getCauses 38, 39  
getChildren 3, 65, 74  
getClassDescription 65  
getClasses 4, 26, 49, 66  
getClassInstances 8, 52, 57, 65, 66, 70  
getClosure 39, 40, 74  
getCorrelationParameters 66, 67, 89  
getEnumVals 67  
getEventCauses 67, 74  
getEventClassName 68  
getEventDescription 68  
getEventExplainedBy 68  
getEventExported 59, 68, 69  
getEventNames 61, 69  
getEvents 61, 67  
getEventSymptoms 69  
getEventType 44, 45, 69, 70  
getEventType\_P 69  
getExplainedBy 3, 40, 68  
getExplains 40, 74  
getf 80  
getf\_T 80  
getf\_t 80  
getfAllProperties 81  
getfAllProperties\_t 81  
getFileno 34  
getfMultipleProperties 81  
getfMultipleProperties\_t 81  
getInstances 3, 5, 6, 26, 33, 48, 49, 70  
getInstrumentationType 70  
getLeafInstances 48, 57, 66, 70  
getLibraries 70, 71  
getMultipleProperties 35, 71, 76, 81  
getMultipleProperties\_t 71  
getObserverId 59, 71  
getOpArgs 48, 71, 72  
getOpArgType 63, 71  
getOpDescription 72  
getOperationArguments 47, 48, 72  
getOperationArgumentType 63, 72  
getOperationDescription 72  
getOperationFlag 72, 73  
getOperationReturnType 72, 73  
getOperations 72  
getOpFlag 72, 73  
getOpNames 72, 73  
getOpReturnType 72, 73  
getParentClass 74  
getProblemClosure 67, 74

getProblemExplanation 74  
getProblemNames 74  
getProblemSymptomState 74  
getPrograms 75  
getPropAccess 75  
getPropDescription 75, 76  
getProperties 76  
getPropertyDescription 75, 76  
getPropertySubscriptionState 78  
getPropertyType 76, 77  
getPropIsReadOnly 76  
getPropIsRelationship 77  
getPropIsRequired 77  
getPropNames 76, 77  
getPropRange 77  
getPropType 76, 77, 78  
getProtocolVersion 34  
getRelatedClass 78  
getRelationNames 78, 79  
getRelations 78  
getRelationTypes 79  
getReverseRelation 79  
getServerName 45  
getSubscriptionState 8, 79  
getThreads 79, 80

## H

hash 5, 6, 18, 19, 20, 21, 22, 62, 88  
hasRequiredProps 81

## I

ICIM 8, 19, 58, 60, 61, 65  
init 4, 6, 7, 8, 25, 29, 30, 43, 49, 50, 52  
insertElement 22, 23, 45, 81  
insertElement\_P 81, 82  
Instance 1, 2, 3, 4, 9, 10, 11, 12, 14, 18, 19, 32, 33, 35, 38, 39, 40, 41, 44, 52, 53, 54, 55, 56, 58, 59, 61, 65, 66, 70, 82  
instanceExists 59, 82  
invoke 3, 17, 19, 22, 23, 36, 37, 44, 54, 58, 82  
invoke\_T 82  
invoke\_t 23, 37, 82, 83  
invokeOperation 37, 49, 82, 83  
invokeOperation\_T 83  
invokeOperation\_t 82, 83  
isAbstract 83  
isBaseOf 84  
isBaseOfOrProxy 84  
isInstrumented 84  
isMember 84

---

isMemberByKey 85	53, 60, 88
isMemberByKeyf 85	Observer 7, 8, 30, 31, 34, 59, 71, 90
isMemberf 85	observer 6, 7, 8, 30, 31, 71
isNull 22, 82	openObjectId papi_Introduction.fm
IsSubscribed 8	2 454639 v
isSubscribed 85	2 469494 v
	2 469498 v
<b>L</b>	2 469501 v
loadLibrary 86	2 469503 v
loadModel 86	2 469512 v
loadProgram 86	2 469526 v
	2 469889 vi
<b>M</b>	2 470472 vi
message URL http	2 470487 vi
//cisco.com/univercd/cc/td/doc/pcat/ XX	2 470517 vi
//tools.cisco.com/RPF/register/register.do XiX	2 471036 v
//www.cisco.com Xvii	2 471125 v
//www.cisco.com/en/US/learning/index.html XXi	2 472125 v
//www.cisco.com/en/US/partner/ordering/index.shtml X	
viii	openObjectId papi_IX.fm
//www.cisco.com/go/iqmagazine XXi	2 10489 xii
//www.cisco.com/go/marketplace/ XX	openObjectId papi_Object.fm
//www.cisco.com/ipj XXi	2 454639 vi
//www.cisco.com/packet XX	2 469490 vi
//www.cisco.com/public/countries_languages.shtml XV	2 469500 vi
ii	2 469511 vi
//www.cisco.com/techsupport xviii	2 469563 vi
//www.cisco.com/techsupport/contacts XiX	2 469564 vi
//www.cisco.com/techsupport/servicerequest XiX	2 469567 vi
//www.cisco.com/univercd/cc/td/doc/es_inpc/pdi.htm	2 469605 vi
xvii	2 469622 vi
//www.cisco.com/univercd/home/home.htm Xvii	2 469625 vi
//www.ciscopress.com XX	2 469635 vi
Model 1, 2, 6, 11, 40, 68, 74, 86	2 469642 vi
	2 469648 vi
<b>N</b>	2 469651 vi
new 4, 7, 19, 25, 27, 29, 30, 50, 52	2 469655 vi
noop 86	2 469659 vi
Notification 7, 8, 9, 10, 11, 12, 13, 14, 15, 41, 86, 90	2 470095 vi
notify 24, 86	2 470948 vi
	openObjectId papi_PREFACE.fm
<b>O</b>	2 435167 v
object 6, 17, 18, 20, 21, 22, 25, 32, 33, 36, 37, 43, 49, 50,	

---

2 436900	v	2 469722	viii
2 438216	v	2 469726	viii
2 438222	v	2 469729	viii
2 675425	v	2 469734	viii
2 675634	v	2 469739	viii
2 675701	v	2 469745	viii
2 675703	v	2 469747	viii
2 675705	v	2 469749	viii
2 675785	v	2 469757	viii
2 675787	v	2 469759	viii
2 675797	v	2 469775	viii
2 675806	v	2 469782	ix
2 675811	v	2 469792	ix
2 675813	v	2 469802	ix
2 675820	v	2 469808	ix
2 675830	v	2 469813	ix
2 675836	v	2 469816	ix
openObjectId papi_Primitives.fm		2 469829	ix
2 454639	viii	2 469837	ix
2 469490	viii	2 469844	ix
2 469500	viii	2 469846	ix
2 469503	viii	2 469851	ix
2 469511	viii	2 469855	ix
2 469536	viii	2 469863	ix
2 469596	viii	2 469871	ix
2 469600	viii	2 469880	ix
2 469618	viii	2 469889	ix
2 469620	viii	2 469894	ix
2 469631	viii	2 469898	ix
2 469646	viii	2 469904	ix
2 469684	viii	2 469908	ix
2 469688	viii	2 469916	ix
2 469694	viii	2 469928	ix
2 469697	viii	2 469932	ix
2 469700	viii	2 469934	ix
2 469706	viii	2 469942	ix
2 469710	viii	2 469946	ix
2 469714	viii	2 469950	ix
2 469718	viii	2 469957	ix

---

2 469964 ix	2 470151 x
2 469968 ix	2 470156 xi
2 469972 ix	2 470165 xi
2 469987 ix	2 470169 xi
2 469989 ix	2 470174 xi
2 469995 ix	2 470176 xi
2 470000 x	2 470181 xi
2 470004 x	2 470187 xi
2 470006 x	2 470194 xi
2 470017 x	2 470205 xi
2 470021 x	2 470210 xi
2 470023 x	2 470215 xi
2 470028 x	2 470219 xi
2 470035 x	2 470225 xi
2 470037 x	2 470229 xi
2 470039 x	2 470239 xi
2 470041 x	2 470245 xi
2 470043 x	2 470247 xi
2 470045 x	2 470249 xi
2 470047 x	2 470263 xi
2 470057 x	2 470277 xi
2 470062 x	2 470282 xi
2 470069 x	2 470288 xi
2 470073 x	2 470294 xi
2 470080 x	2 470298 xi
2 470086 x	2 470306 xi
2 470090 x	2 470314 xi
2 470094 x	2 470323 xi
2 470098 x	2 470333 xi
2 470109 x	2 470337 xi
2 470114 x	2 470342 xi
2 470118 x	2 470344 xi
2 470120 x	2 470348 xi
2 470123 x	2 470352 xi
2 470125 x	2 470354 xi
2 470133 x	2 470356 xii
2 470139 x	2 470360 xii
2 470143 x	2 470370 xii
2 470145 x	2 470374 xii

- 2 470381 xii
- 2 470383 xii
- 2 470403 xii
- 2 470407 xii
- 2 470417 xii
- 2 470421 xii
- 2 470425 xii
- 2 470431 xii
- 2 470433 xii
- 2 470438 xii
- 2 470443 xii
- 2 470447 xii
- 2 470449 xii
- 2 470452 xii
- 2 470456 xii
- 2 470460 xii
- 2 470464 xii
- 2 470474 xii
- 2 470476 xii
- 2 476403 viii
- 2 476478 viii
- 2 478542 viii
- openObjectId papi\_Session.fm
  - 2 454639 vi
  - 2 469490 vi
  - 2 469500 vi
  - 2 469515 vi
  - 2 469519 vi
  - 2 469521 vi
  - 2 469523 vi
  - 2 469526 vi
  - 2 469528 vi
  - 2 469537 vii
  - 2 469539 vii
  - 2 469565 vii
  - 2 469582 vii
  - 2 469585 vii
  - 2 469591 vii
  - 2 469596 vii
- 2 469604 vii
- 2 469609 vii
- 2 469629 vii
- 2 469634 vii
- 2 469645 vii
- 2 469646 vii
- 2 469657 vii
- 2 469662 vii
- 2 469673 vii
- 2 469675 vii
- 2 469678 vii
- 2 469701 vii
- 2 469722 vii
- 2 469735 vii
- 2 469750 vii
- 2 469763 vii
- 2 469774 vii
- 2 469785 vii
- 2 469815 vii
- 2 469853 vii
- 2 469861 vii
- 2 469873 vii
- 2 469876 vii
- 2 469881 vii
- 2 472952 vi
- 2 472972 vii
- 2 473478 vii
- Operation 1, 2, 3, 4, 5, 18, 19, 23, 24, 33, 36, 37, 44, 47, 48, 49, 51, 58, 62, 63, 72, 73, 82
- P
  - Perl xiii, xiv, 1, 3, 4, 5, 6, 7, 9, 10, 14, 20, 22, 26, 27, 28, 32, 34, 37, 43, 47, 48, 49, 53, 62
  - Perlfunc 50
  - ping 86
  - Primitive xiv, 3, 4, 5, 26, 27, 28, 29, 33, 34, 35, 36, 37, 44, 45, 47, 48, 49, 51, 52, 54, 56, 67, 69, 70, 71, 74, 76, 81, 82, 88, 89
  - primitiveIsAvailable 35
  - Property 2, 3, 4, 5, 8, 12, 14, 18, 19, 20, 21, 22, 33, 36, 48, 49, 56, 60, 61, 62, 63, 64, 67, 71, 75, 76, 77, 78, 79, 80, 81, 86, 87, 89



---

propertySubscribe 6, 8, 19, 86, 87  
propertySubscribeAll 87  
propertyUnsubscribe 8, 87  
propertyUnsubscribeAll 87  
purgeObserver 58, 87  
put 17, 18, 22, 34, 35, 36, 49, 88  
put\_P 36, 87, 88

## Q

quit 88, 89

## R

reattach 30, 31  
receiveEvent 6, 8, 9, 15, 31  
Relationship 1, 2, 12, 22, 23, 24, 36, 38, 39, 44, 45, 57, 58,  
77, 78, 79, 84, 85, 88  
relationship 2, 84  
Relationshipset 2, 81  
removeElement 22, 23, 24, 45, 88  
removeElement\_P 88, 89  
removeElementByKey 89  
restoreRepository 89

## S

save 35, 89, 90  
setCorrelationParameters 89  
shutdown 88, 89  
SM\_AUTHORITY 28, 29  
SM\_HOME 29, 90  
storeAllRepository 26, 89  
storeClassRepository 90  
Subscribe 8, 31, 41, 42  
subscribe 8, 19, 41, 42, 87, 90  
subscribeAll 8, 90  
subscribeEvent 90, 92  
Subscription xiii, 6, 7, 8, 13, 14, 30, 31, 42, 78, 79, 87, 90,  
91

## T

Timestamp 9, 10, 11, 12, 13, 14, 15  
topologySubscribe 8, 90, 91  
topologyUnsubscribe 8, 90, 91  
transaction 42, 43, 91  
transactionAbort 91  
transactionCommit 91  
transactionStart 91

## U

Unsubscribe 8, 41  
unsubscribe 8, 41, 42, 92  
unsubscribeAll 8, 92  
unsubscribeEvent 90, 92

