Open Source Used In NSO NED: odl-conroller

Cisco Systems, Inc.
www.cisco.com

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

Text Part Number: 78EE117C99-111017988

This document contains licenses and notices for open source software used in this product. With respect to the free/open source software listed in this document, if you have any questions please contact us at external-opensource-requests@cisco.com.

In your requests please include the following reference number 78EE117C99-111017988

Contents

1.1 json 20140107

1.1.1 Available under license :

/*

Copyright (c) 2002 JSON.org

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

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

The Software shall be used for Good, not Evil.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

*/

1.2 Staxon-core 1.3

1.2.1 Available under license :

```
/*
* Copyright 2011, 2012 Odysseus Software GmbH
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package de.odysseus.staxon.event;

import java.io.IOException;
import java.io.StringWriter;
import java.io.Writer;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import javax.xml.XMLConstants;
import javax.xml.namespace.NamespaceContext;
import javax.xml.namespace.QName;
import javax.xml.stream.Location;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.Comment;
import javax.xml.stream.events.EndDocument;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.EntityDeclaration;
import javax.xml.stream.events.EntityReference;
import javax.xml.stream.events.Namespace;
import javax.xml.stream.events.ProcessingInstruction;
import javax.xml.stream.events.StartDocument;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;
```

```
import javax.xml.stream.util.XMLEventAllocator;
import javax.xml.stream.util.XMLEventConsumer;

/**
* Simple implementation of {@link XMLEventAllocator}.
*/
public class SimpleXMLEventAllocator implements XMLEventAllocator {
 static abstract class AbstractXMLEvent implements XMLEvent {
  static <E> List<E> toList(Iterator<E> iterator) {
   if (iterator == null || !iterator.hasNext()) {
    return Collections.emptyList();
   } else {
    List<E> list = new ArrayList<E>();
    while (iterator.hasNext()) {
     list.add(iterator.next());
    }
    return list;
   }
  }

  final int eventType;
  final Location location;
  AbstractXMLEvent(int eventType, Location location) {
   this.eventType = eventType;
   this.location = location;
  }
  @Override
  public Characters asCharacters() {
   return (Characters) this;
  }
  @Override
  public StartElement asStartElement() {
   return (StartElement) this;
  }
  @Override
  public EndElement asEndElement() {
   return (EndElement) this;
  }
  @Override
  public int getEventType() {
   return eventType;
  }
  @Override
  public Location getLocation() {
   return location;
  }
  @Override
  public QName getSchemaType() {
```

```
 return null;
}
@Override
public boolean isAttribute() {
 return eventType == XMLStreamConstants.ATTRIBUTE;
}
@Override
public boolean isCharacters() {
 return eventType == XMLStreamConstants.CHARACTERS || eventType == XMLStreamConstants.CDATA;
}
@Override
public boolean isEndDocument() {
 return eventType == XMLStreamConstants.END_DOCUMENT;
}
@Override
public boolean isEndElement() {
 return eventType == XMLStreamConstants.END_ELEMENT;
}
@Override
public boolean isEntityReference() {
 return eventType == XMLStreamConstants.ENTITY_REFERENCE;
}
@Override
public boolean isNamespace() {
 return eventType == XMLStreamConstants.NAMESPACE;
}
@Override
public boolean isProcessingInstruction() {
 return eventType == XMLStreamConstants.PROCESSING_INSTRUCTION;
}
@Override
public boolean isStartDocument() {
 return eventType == XMLStreamConstants.START_DOCUMENT;
}
@Override
public boolean isStartElement() {
 return eventType == XMLStreamConstants.START_ELEMENT;
}
@Override
public String toString() {
 try {
  Writer writer = new StringWriter();
  writer.write(getClass().getSimpleName());
  writer.write('(');
  writeAsEncodedUnicodeInternal(writer);
  writer.write(')');
  return writer.toString();
 } catch (IOException e) {
```

```
  return super.toString();
 }
}
@Override
public void writeAsEncodedUnicode(Writer writer) throws XMLStreamException {
 try {
  writeAsEncodedUnicodeInternal(writer);
 } catch (IOException e) {
  throw new XMLStreamException(e);
 }

}
abstract void writeAsEncodedUnicodeInternal(Writer writer) throws IOException;
}

static class AttributeEvent extends AbstractXMLEvent implements Attribute {
 final QName name;
 final String value;
 final boolean specified;
 AttributeEvent(Location location, QName name, String value, boolean specified) {
  this(XMLStreamConstants.ATTRIBUTE, location, name, value, specified);
 }
 AttributeEvent(int eventType, Location location, QName name, String value, boolean specified) {
  super(eventType, location);
  assert eventType == XMLStreamConstants.ATTRIBUTE || eventType == XMLStreamConstants.NAMESPACE;
  this.name = name;
  this.value = value;
  this.specified = specified;
 }
 @Override
 public String getDTDType() {
  return "CDATA";
 }
 @Override
 public QName getName() {
  return name;
 }
 @Override
 public String getValue() {
  return value;
 }
 @Override
 public boolean isSpecified() {
  return specified;
 }
 @Override
 void writeAsEncodedUnicodeInternal(Writer writer) throws IOException {
  if (!XMLConstants.DEFAULT_NS_PREFIX.equals(name.getPrefix())) {
```

```
   writer.write(name.getPrefix());
   writer.write(':');
  }
  writer.write(name.getLocalPart());
  writer.write('=');
  writer.write("");
  for (int i = 0; i < value.length(); i++) {
   char c = value.charAt(i);
   switch (c) {
   case '<':
    writer.write("&lt;");
    break;
   case '>':
    writer.write("&gt;");
    break;
   case '&':
    writer.write("&amp;");
    break;
   case "":
    writer.write("&quot;");
    break;
   default:
    writer.write(c);
   }
  }
  writer.write("");
 }
}

static class CharactersEvent extends AbstractXMLEvent implements Characters {
 final String data;
 final boolean whitespace;
 CharactersEvent(XMLStreamReader reader) {
  this(reader.getEventType(), reader.getLocation(), reader.getText(), reader.isWhiteSpace());
 }
 CharactersEvent(int eventType, Location location, String data, boolean whitespace) {
  super(eventType, location);
  assert eventType == XMLStreamConstants.CHARACTERS
    || eventType == XMLStreamConstants.CDATA
    || eventType == XMLStreamConstants.SPACE;
  this.data = data;
  this.whitespace = whitespace;
 }
 @Override
 public String getData() {
  return data;
 }
 @Override
```

```
 public boolean isCData() {
  return eventType == XMLStreamConstants.CDATA;
 }
 @Override
 public boolean isIgnorableWhiteSpace() {
  return eventType == XMLStreamConstants.SPACE;
 }
 @Override
 public boolean isWhiteSpace() {
  return whitespace;
 }
 @Override
 void writeAsEncodedUnicodeInternal(Writer writer) throws IOException {
  if (isCData()) {
   writer.write("<![CDATA[");
   writer.write(data);
   writer.write("]]>");
  } else if (!isIgnorableWhiteSpace()) { // API doc: No indentation or whitespace should be "outputted".
   for (int i = 0; i < data.length(); i++) {
    char c = data.charAt(i);
    switch (c) {
    case '<':
     writer.write("&lt;");
     break;
    case '>':
     writer.write("&gt;");
     break;
    case '&':
     writer.write("&amp;");
     break;
    default:
     writer.write(c);
    }
   }
  }
 }
}

static class CommentEvent extends AbstractXMLEvent implements Comment {
 final String text;
 CommentEvent(XMLStreamReader reader) {
  super(reader.getEventType(), reader.getLocation());
  assert eventType == XMLStreamConstants.COMMENT;
  text = reader.getText();
 }
 CommentEvent(Location location, String text) {
  super(XMLStreamConstants.COMMENT, location);
  this.text = text;
```

```
 }
 @Override
 public String getText() {
  return text;
 }
 @Override
 void writeAsEncodedUnicodeInternal(Writer writer) throws IOException {
  writer.write("<!--");
  writer.write(text);
  writer.write("-->");
 }
}

static class EndDocumentEvent extends AbstractXMLEvent implements EndDocument {
 EndDocumentEvent(XMLStreamReader reader) {
  super(reader.getEventType(), reader.getLocation());
  assert eventType == XMLStreamConstants.END_DOCUMENT;
 }
 EndDocumentEvent(Location location) {
  super(XMLStreamConstants.END_DOCUMENT, location);
 }
 @Override
 void writeAsEncodedUnicodeInternal(Writer writer) {
  // silence
 }
}

static class EndElementEvent extends AbstractXMLEvent implements EndElement {
 final QName name;
 final List<NamespaceEvent> namespaces;
 EndElementEvent(XMLStreamReader reader) {
  super(reader.getEventType(), reader.getLocation());
  assert eventType == XMLStreamConstants.END_ELEMENT;
  name = reader.getName();
  if (reader.getNamespaceCount() == 0) {
   namespaces = Collections.emptyList();
  } else {
   namespaces = new ArrayList<NamespaceEvent>(reader.getNamespaceCount());
   for (int i = 0; i < reader.getNamespaceCount(); i++) {
    namespaces.add(new NamespaceEvent(location, reader.getNamespaceURI(i), reader.getNamespacePrefix(i)));
   }
  }
 }
 EndElementEvent(Location location, QName name, Iterator<NamespaceEvent> namespaces) {
  super(XMLStreamConstants.END_ELEMENT, location);
  this.name = name;
  this.namespaces = toList(namespaces);
 }
```

```
 @Override
 public QName getName() {
  return name;
 }
 @Override
 public Iterator<?> getNamespaces() {
  return namespaces.iterator();
 }
 @Override
 void writeAsEncodedUnicodeInternal(Writer writer) throws IOException {
  writer.write("</");
  if (!XMLConstants.DEFAULT_NS_PREFIX.equals(name.getPrefix())) {
   writer.write(name.getPrefix());
   writer.write(':');
  }
  writer.write(name.getLocalPart());
  writer.write('>');
 }
}

static class EntityReferenceEvent extends AbstractXMLEvent implements EntityReference {
 final String name;
 final EntityDeclaration declaration;
 EntityReferenceEvent(XMLStreamReader reader) {
  super(reader.getEventType(), reader.getLocation());
  assert eventType == XMLStreamConstants.ENTITY_REFERENCE;
  name = reader.getText();
  declaration = null; // TODO
 }
 EntityReferenceEvent(Location location, String name, EntityDeclaration declaration) {
  super(XMLStreamConstants.ENTITY_REFERENCE, location);
  this.name = name;
  this.declaration = declaration;
 }
 @Override
 public EntityDeclaration getDeclaration() {
  return declaration;
 }
 @Override
 public String getName() {
  return name;
 }
 @Override
 void writeAsEncodedUnicodeInternal(Writer writer) throws IOException {
  writer.write('&');
  writer.write(name);
  writer.write(';');
 }
```

```
 }

 static class NamespaceEvent extends AttributeEvent implements Namespace {
  static QName createName(String prefix) {
   if (prefix == null || XMLConstants.DEFAULT_NS_PREFIX.equals(prefix)) {
    return new QName(XMLConstants.XMLNS_ATTRIBUTE_NS_URI, XMLConstants.XMLNS_ATTRIBUTE);
   } else {
    return new QName(XMLConstants.XMLNS_ATTRIBUTE_NS_URI, prefix,
XMLConstants.XMLNS_ATTRIBUTE);
   }
  }
  NamespaceEvent(Location location, String namespaceURI, String prefix) {
   super(XMLStreamConstants.NAMESPACE, location, createName(prefix), namespaceURI, true);
  }
  @Override
  public String getPrefix() {
   return isDefaultNamespaceDeclaration() ? XMLConstants.DEFAULT_NS_PREFIX : getName().getLocalPart();
  }
  @Override
  public String getNamespaceURI() {
   return getValue();
  }
  @Override
  public boolean isDefaultNamespaceDeclaration() {
   return XMLConstants.DEFAULT_NS_PREFIX.equals(getName().getPrefix());
  }
 }

 static class ProcessingInstructionEvent extends AbstractXMLEvent implements ProcessingInstruction {
  final String target;
  final String data;
  ProcessingInstructionEvent(XMLStreamReader reader) {
   super(reader.getEventType(), reader.getLocation());
   assert eventType == XMLStreamConstants.PROCESSING_INSTRUCTION;
   target = reader.getPITarget();
   data = reader.getPIData();
  }
  ProcessingInstructionEvent(Location location, String target, String data) {
   super(XMLStreamConstants.PROCESSING_INSTRUCTION, location);
   this.target = target;
   this.data = data;
  }
  @Override
  public String getTarget() {
   return target;
  }
  @Override
  public String getData() {
```

```
 return data;
 }
 @Override
 void writeAsEncodedUnicodeInternal(Writer writer) throws IOException {
  writer.write("<?");
  writer.write(target);
  if (data != null) {
   writer.write(' ');
   writer.write(data.trim());
  }
  writer.write("?>");
 }
}

static class StartDocumentEvent extends AbstractXMLEvent implements StartDocument {
 final String encodingScheme;
 final String version;
 final Boolean standalone;
 StartDocumentEvent(XMLStreamReader reader) {
  super(reader.getEventType(), reader.getLocation());
  assert eventType == XMLStreamConstants.START_DOCUMENT;
  encodingScheme = reader.getCharacterEncodingScheme();
  version = reader.getVersion() == null ? "1.0" : reader.getVersion();
  standalone = reader.standaloneSet() ? Boolean.valueOf(reader.isStandalone()) : null;
 }
 StartDocumentEvent(Location location, String encoding, String version, Boolean standalone) {
  super(XMLStreamConstants.START_DOCUMENT, location);
  this.encodingScheme = encoding;
  this.version = version == null ? "1.0" : version;
  this.standalone = standalone;
 }
 @Override
 public String getCharacterEncodingScheme() {
  return encodingSet() ? encodingScheme : "UTF-8";
 }
 @Override
 public boolean encodingSet() {
  return encodingScheme != null;
 }
 @Override
 public String getVersion() {
  return version;
 }
 @Override
 public boolean isStandalone() {
  return standaloneSet() ? standalone.booleanValue() : false;
 }
 @Override
```

```
 public boolean standaloneSet() {
  return standalone != null;
 }
 @Override
 public String getSystemId() {
  return location.getSystemId();
 }
 @Override
 void writeAsEncodedUnicodeInternal(Writer writer) throws IOException {
  writer.write("<?xml version=\"");
  writer.write(version);
  writer.write("");
  if (encodingSet()) {
   writer.write(" encoding=\"");
   writer.write(encodingScheme);
   writer.write("");
  }
  if (standaloneSet()) {
   writer.write(" standalone=\"");
   writer.write(standalone ? "yes" : "no");
   writer.write("");
  }
  writer.write("?>");
 }
}

 static class StartElementEvent extends AbstractXMLEvent implements StartElement {
  final QName name;
  final List<AttributeEvent> attributes;
  final List<NamespaceEvent> namespaces;
  final NamespaceContext context;
  StartElementEvent(XMLStreamReader reader) {
   super(reader.getEventType(), reader.getLocation());
   assert eventType == XMLStreamConstants.START_ELEMENT;
   name = reader.getName();
   context = reader.getNamespaceContext();
   if (reader.getAttributeCount() == 0) {
    attributes = Collections.emptyList();
   } else {
    attributes = new ArrayList<AttributeEvent>(reader.getAttributeCount());
    for (int i = 0; i < reader.getAttributeCount(); i++) {
     attributes.add(new AttributeEvent(location, reader.getAttributeName(i), reader.getAttributeValue(i),
reader.isAttributeSpecified(i)));
    }
   }
   if (reader.getNamespaceCount() == 0) {
    namespaces = Collections.emptyList();
   } else {
```

```
  namespaces = new ArrayList<NamespaceEvent>(reader.getNamespaceCount());
  for (int i = 0; i < reader.getNamespaceCount(); i++) {
   namespaces.add(new NamespaceEvent(location, reader.getNamespaceURI(i), reader.getNamespacePrefix(i)));
  }
 }
}
 StartElementEvent(Location location, QName name, Iterator<AttributeEvent> attributes,
Iterator<NamespaceEvent> namespaces, NamespaceContext context) {
 super(XMLStreamConstants.START_ELEMENT, location);
 this.name = name;
 this.context = context;
 this.attributes = toList(attributes);
 this.namespaces = toList(namespaces);
}
@Override
public Attribute getAttributeByName(QName name) {
 for (Attribute attribute : attributes) {
  if (attribute.getName().equals(name)) {
   return attribute;
  }
 }
 return null;
}
@Override
public Iterator<?> getAttributes() {
 return attributes.iterator();
}
@Override
public QName getName() {
 return name;
}
@Override
public NamespaceContext getNamespaceContext() {
 return context;
}
@Override
public Iterator<?> getNamespaces() {
 return namespaces.iterator();
}
@Override
public String getNamespaceURI(String prefix) {
 for (Namespace namespace : namespaces) {
  if (namespace.getPrefix().equals(prefix)) {
   return namespace.getNamespaceURI();
  }
 }
 return null;
}
```

```
 @Override
 void writeAsEncodedUnicodeInternal(Writer writer) throws IOException {
  writer.write('<');
  if (!XMLConstants.DEFAULT_NS_PREFIX.equals(name.getPrefix())) {
   writer.write(name.getPrefix());
   writer.write(':');
  }
  writer.write(name.getLocalPart());
  for (NamespaceEvent namespace : namespaces) {
   writer.write(' ');
   namespace.writeAsEncodedUnicodeInternal(writer);
  }
  for (AttributeEvent attribute : attributes) {
   if (attribute.isSpecified()) {
    writer.write(' ');
    attribute.writeAsEncodedUnicodeInternal(writer);
   }
  }
  writer.write('>');
 }
}

@Override
public XMLEventAllocator newInstance() {
 return this;
}

@Override
public XMLEvent allocate(XMLStreamReader reader) throws XMLStreamException {
 switch (reader.getEventType()) {
 case XMLStreamConstants.CDATA:
 case XMLStreamConstants.CHARACTERS:
 case XMLStreamConstants.SPACE:
  return new CharactersEvent(reader);
 case XMLStreamConstants.COMMENT:
  return new CommentEvent(reader);
 case XMLStreamConstants.DTD:
  throw new UnsupportedOperationException();
 case XMLStreamConstants.END_DOCUMENT:
  return new EndDocumentEvent(reader);
 case XMLStreamConstants.END_ELEMENT:
  return new EndElementEvent(reader);
 case XMLStreamConstants.ENTITY_REFERENCE:
  return new EntityReferenceEvent(reader);
 case XMLStreamConstants.NOTATION_DECLARATION:
  throw new UnsupportedOperationException();
 case XMLStreamConstants.PROCESSING_INSTRUCTION:
  return new ProcessingInstructionEvent(reader);
```

```
  case XMLStreamConstants.START_DOCUMENT:
   return new StartDocumentEvent(reader);
  case XMLStreamConstants.START_ELEMENT:
   return new StartElementEvent(reader);
  default:
   throw new XMLStreamException("Unexpected event type: " + reader.getEventType());
  }
 }

 @Override
 public void allocate(XMLStreamReader reader, XMLEventConsumer consumer) throws XMLStreamException {
  consumer.add(allocate(reader));
 }
}
/*
* Copyright 2011, 2012 Odysseus Software GmbH
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package de.odysseus.staxon.event;

import java.util.Iterator;

import javax.xml.XMLConstants;
import javax.xml.namespace.NamespaceContext;
import javax.xml.namespace.QName;
import javax.xml.stream.Location;
import javax.xml.stream.XMLEventFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.Comment;
import javax.xml.stream.events.DTD;
import javax.xml.stream.events.EndDocument;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.EntityDeclaration;
import javax.xml.stream.events.EntityReference;
import javax.xml.stream.events.Namespace;
```

```
import javax.xml.stream.events.ProcessingInstruction;
import javax.xml.stream.events.StartDocument;
import javax.xml.stream.events.StartElement;

/**
* Simple implementation of {@link XMLEventFactory}.
*/
public class SimpleXMLEventFactory extends XMLEventFactory {
 private Location location;

 public SimpleXMLEventFactory() {
  this(null);
 }

 public SimpleXMLEventFactory(Location location) {
  setLocation(location);
 }

 @Override
 public void setLocation(Location location) {
  this.location = location;
 }

 @Override
 public Attribute createAttribute(String prefix, String namespaceURI, String localName, String value) {
  return createAttribute(new QName(namespaceURI, localName, prefix), value);
 }

 @Override
 public Attribute createAttribute(String localName, String value) {
  return createAttribute(new QName(localName), value);
 }

 @Override
 public Attribute createAttribute(QName name, String value) {
  return new SimpleXMLEventAllocator.AttributeEvent(location, name, value, true);
 }

 @Override
 public Namespace createNamespace(String namespaceURI) {
  return createNamespace(XMLConstants.DEFAULT_NS_PREFIX, namespaceURI);
 }

 @Override
 public Namespace createNamespace(String prefix, String namespaceURI) {
  return new SimpleXMLEventAllocator.NamespaceEvent(location, namespaceURI, prefix);
 }
```

```
@SuppressWarnings({"rawtypes", "unchecked"})
StartElement createStartElement(QName name, Iterator attributes, Iterator namespaces, NamespaceContext context)
{
 return new SimpleXMLEventAllocator.StartElementEvent(location, name, attributes, namespaces, context);
 }

 @Override
 @SuppressWarnings({"rawtypes"})
 public StartElement createStartElement(QName name, Iterator attributes, Iterator namespaces) {
  return createStartElement(name, attributes, namespaces, null);
 }

 @Override
 public StartElement createStartElement(String prefix, String namespaceUri, String localName) {
  return createStartElement(new QName(namespaceUri, localName, prefix), null, null);
 }

 @Override
 @SuppressWarnings({"rawtypes"})
 public StartElement createStartElement(String prefix, String namespaceUri, String localName, Iterator attributes,
Iterator namespaces) {
  return createStartElement(new QName(namespaceUri, localName, prefix), attributes, namespaces);
 }

 @Override
 @SuppressWarnings({"rawtypes"})
 public StartElement createStartElement(String prefix, String namespaceUri, String localName, Iterator attributes,
Iterator namespaces, NamespaceContext context) {
  return createStartElement(new QName(namespaceUri, localName, prefix), attributes, namespaces, context);
 }

 @Override
 @SuppressWarnings({"rawtypes", "unchecked"})
 public EndElement createEndElement(QName name, Iterator namespaces) {
  return new SimpleXMLEventAllocator.EndElementEvent(location, name, namespaces);
 }

 @Override
 public EndElement createEndElement(String prefix, String namespaceUri, String localName) {
  return createEndElement(new QName(namespaceUri, localName, prefix), null);
 }

 @Override
 @SuppressWarnings({"rawtypes"})
 public EndElement createEndElement(String prefix, String namespaceUri, String localName, Iterator namespaces) {
  return createEndElement(new QName(namespaceUri, localName, prefix), namespaces);
 }
```

```
 @Override
 public Characters createCharacters(String content) {
  return new SimpleXMLEventAllocator.CharactersEvent(XMLStreamConstants.CHARACTERS, location, content,
false);
 }

 @Override
 public Characters createCData(String content) {
  return new SimpleXMLEventAllocator.CharactersEvent(XMLStreamConstants.CDATA, location, content, false);
 }

 @Override
 public Characters createSpace(String content) {
  return new SimpleXMLEventAllocator.CharactersEvent(XMLStreamConstants.CHARACTERS, location, content,
true);
 }

 @Override
 public Characters createIgnorableSpace(String content) {
  return new SimpleXMLEventAllocator.CharactersEvent(XMLStreamConstants.SPACE, location, content, true);
 }

 @Override
 public StartDocument createStartDocument() {
  return new SimpleXMLEventAllocator.StartDocumentEvent(location, null, null, null);
 }

 @Override
 public StartDocument createStartDocument(String encoding, String version, boolean standalone) {
  return new SimpleXMLEventAllocator.StartDocumentEvent(location, encoding, version, standalone);
 }

 @Override
 public StartDocument createStartDocument(String encoding, String version) {
  return new SimpleXMLEventAllocator.StartDocumentEvent(location, encoding, version, null);
 }

 @Override
 public StartDocument createStartDocument(String encoding) {
  return new SimpleXMLEventAllocator.StartDocumentEvent(location, encoding, null, null);
 }

 @Override
 public EndDocument createEndDocument() {
  return new SimpleXMLEventAllocator.EndDocumentEvent(location);
 }

 @Override
```

```
 public EntityReference createEntityReference(String name, EntityDeclaration declaration) {
  return new SimpleXMLEventAllocator.EntityReferenceEvent(location, name, declaration);
 }

 @Override
 public Comment createComment(String text) {
  return new SimpleXMLEventAllocator.CommentEvent(location, text);
 }

 @Override
 public ProcessingInstruction createProcessingInstruction(String target, String data) {
  return new SimpleXMLEventAllocator.ProcessingInstructionEvent(location, target, data);
 }

 @Override
 public DTD createDTD(String dtd) {
  throw new UnsupportedOperationException("DTD event is not supported");
 }
}
/*
* Copyright 2011, 2012 Odysseus Software GmbH
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package de.odysseus.staxon.event;

import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.events.XMLEvent;
import javax.xml.stream.util.XMLEventAllocator;

/**
* Simple implementation of {@link XMLEventReader}.
*/
public class SimpleXMLEventReader implements XMLEventReader {
 private final XMLEventAllocator allocator;
```

```
private final XMLStreamReader delegate;

private int currentEventType = -1;
private XMLEvent peekedEvent = null;

public SimpleXMLEventReader(XMLStreamReader delegate) {
 this(delegate, new SimpleXMLEventAllocator());
}

public SimpleXMLEventReader(XMLStreamReader delegate, XMLEventAllocator allocator) {
 this.delegate = delegate;
 this.allocator = allocator;
}

protected XMLEvent allocate() throws XMLStreamException {
 return allocator.allocate(delegate);
}

@Override
public void close() throws XMLStreamException {
 delegate.close();
}

@Override
public String getElementText() throws XMLStreamException {
 if (currentEventType != XMLStreamConstants.START_ELEMENT) {
  throw new XMLStreamException("Expected start element event");
 }

 if (peekedEvent == null) {
  String result = delegate.getElementText();
  currentEventType = delegate.getEventType();
  if (currentEventType != XMLStreamConstants.END_ELEMENT) {
   throw new XMLStreamException("Expected end element event");
  }
  return result;
 }

 currentEventType = peekedEvent.getEventType();
 assert currentEventType == delegate.getEventType();
 peekedEvent = null;

 String leadText = null;
 switch (currentEventType) {
 case XMLStreamConstants.CDATA:
 case XMLStreamConstants.CHARACTERS:
 case XMLStreamConstants.ENTITY_REFERENCE:
  leadText = delegate.getText();
```

```
 break;
case XMLStreamConstants.COMMENT:
case XMLStreamConstants.PROCESSING_INSTRUCTION:
case XMLStreamConstants.SPACE:
 break;
case XMLStreamConstants.END_ELEMENT:
 return "";
default:
 throw new XMLStreamException("Unexpected event type " + currentEventType, delegate.getLocation());
}

StringBuilder builder = null;
while (true) {
 currentEventType = delegate.next();
 switch (currentEventType) {
 case XMLStreamConstants.CDATA:
 case XMLStreamConstants.CHARACTERS:
 case XMLStreamConstants.ENTITY_REFERENCE:
  if (leadText == null) { // first event?
   leadText = delegate.getText();
  } else {
   if (builder == null) { // second event?
    builder = new StringBuilder(leadText);
   }
   builder.append(delegate.getText());
  }
  break;
 case XMLStreamConstants.COMMENT:
 case XMLStreamConstants.PROCESSING_INSTRUCTION:
 case XMLStreamConstants.SPACE:
  break;
 case XMLStreamConstants.END_ELEMENT:
  return builder == null ? (leadText == null ? "" : leadText) : builder.toString();
 default:
  throw new XMLStreamException("Unexpected event type " + currentEventType, delegate.getLocation());
 }
}
}

@Override
public Object getProperty(String name) {
 return delegate.getProperty(name);
}


@Override
public boolean hasNext() {
 try {
```

```
 return peek() != null;
 } catch (XMLStreamException e) {
  throw new RuntimeException("Cannot determine next state", e);
 }
}

@Override
public XMLEvent nextEvent() throws XMLStreamException {
 XMLEvent currentEvent = peek();
 if (currentEvent == null) {
  throw new XMLStreamException("no more events");
 }
 currentEventType = currentEvent.getEventType();
 peekedEvent = null;
 return currentEvent;
}

@Override
public Object next() {
 try {
  return nextEvent();
 } catch (XMLStreamException e) {
  throw new RuntimeException(e);
 }
}

@Override
public XMLEvent nextTag() throws XMLStreamException {
 if (peekedEvent == null) {
  currentEventType = delegate.nextTag();
  if (currentEventType != XMLStreamConstants.START_ELEMENT && currentEventType !=
XMLStreamConstants.END_ELEMENT) {
   throw new XMLStreamException("Expected start element event or end element event");

  }
  return allocate();
 }

 currentEventType = peekedEvent.getEventType();
 assert currentEventType == delegate.getEventType();
 XMLEvent event = peekedEvent;
 peekedEvent = null;

 switch (event.getEventType()) {
 case XMLStreamConstants.START_DOCUMENT:
  break;
 case XMLStreamConstants.COMMENT:
 case XMLStreamConstants.PROCESSING_INSTRUCTION:
```

```
 case XMLStreamConstants.SPACE:
  break;
 case XMLStreamConstants.CDATA:
 case XMLStreamConstants.CHARACTERS:
  if (!event.asCharacters().isWhiteSpace()) {
   throw new XMLStreamException("Encountered non-whitespace text");
  }
  break;
 case XMLStreamConstants.START_ELEMENT:
 case XMLStreamConstants.END_ELEMENT:
  return event;
 default:
  throw new XMLStreamException("Encountered unexpected event: " + event.getEventType());
 }

 while (true) {
  currentEventType = delegate.next();
  switch (currentEventType) {
  case XMLStreamConstants.COMMENT:
  case XMLStreamConstants.PROCESSING_INSTRUCTION:
  case XMLStreamConstants.SPACE:
   continue;
  case XMLStreamConstants.CDATA:
  case XMLStreamConstants.CHARACTERS:
   if (!delegate.isWhiteSpace()) {
    throw new XMLStreamException("Encountered non-whitespace text");
   }
   continue;
  case XMLStreamConstants.START_ELEMENT:
  case XMLStreamConstants.END_ELEMENT:
   return allocate();
  default:
   throw new XMLStreamException("Encountered unexpected event: " + delegate.getEventType());
  }
 }
}

@Override
public XMLEvent peek() throws XMLStreamException {
 if (peekedEvent == null) {
  if (currentEventType < 0) { // first event
   peekedEvent = allocate();
  } else if (delegate.hasNext()) {
   delegate.next();
   peekedEvent = allocate();
  }
 }
 return peekedEvent;
```

```
 }

 @Override
 public void remove() {
  throw new UnsupportedOperationException("Cannot remove events");
 }
}
/*
* Copyright 2011, 2012 Odysseus Software GmbH
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package de.odysseus.staxon.event;

import java.util.Iterator;

import javax.xml.namespace.NamespaceContext;
import javax.xml.namespace.QName;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLEventWriter;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.Comment;
import javax.xml.stream.events.DTD;
import javax.xml.stream.events.EntityReference;
import javax.xml.stream.events.Namespace;
import javax.xml.stream.events.ProcessingInstruction;
import javax.xml.stream.events.StartDocument;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;

/**
* Simple implementation of {@link XMLEventWriter}.
*/
public class SimpleXMLEventWriter implements XMLEventWriter {
```

```
 private final XMLStreamWriter delegate;

 public SimpleXMLEventWriter(XMLStreamWriter delegate) {
  this.delegate = delegate;
 }

 @Override
 public void add(XMLEvent event) throws XMLStreamException {
  switch (event.getEventType()) {
  case XMLStreamConstants.ATTRIBUTE:
   Attribute attribute = (Attribute) event;
   QName attrName = attribute.getName();
   delegate.writeAttribute(attrName.getPrefix(), attrName.getNamespaceURI(), attrName.getLocalPart(),
attribute.getValue());
   break;
  case XMLStreamConstants.END_DOCUMENT:
   delegate.writeEndDocument();
   break;
  case XMLStreamConstants.END_ELEMENT:
   delegate.writeEndElement();
   break;
  case XMLStreamConstants.NAMESPACE:
   Namespace namespace = (Namespace) event;
   delegate.writeNamespace(namespace.getPrefix(), namespace.getNamespaceURI());
   break;
  case XMLStreamConstants.START_DOCUMENT:
   StartDocument startDocument = (StartDocument) event;
   if (startDocument.encodingSet()) { // encoding defined?
    delegate.writeStartDocument(startDocument.getCharacterEncodingScheme(), startDocument.getVersion());
   } else {
    delegate.writeStartDocument(startDocument.getVersion());
   }
   break;
  case XMLStreamConstants.START_ELEMENT:
   StartElement startElement = event.asStartElement();
   QName elemName = startElement.getName();
   delegate.writeStartElement(elemName.getPrefix(), elemName.getLocalPart(), elemName.getNamespaceURI());
   Iterator<?> namespaces = startElement.getNamespaces();
   while (namespaces.hasNext()) {
    add((Namespace)namespaces.next());
   }
   Iterator<?> attributes = startElement.getAttributes();
   while (attributes.hasNext()) {
    add((Attribute)attributes.next());
   }
   break;
  case XMLStreamConstants.CHARACTERS:
  case XMLStreamConstants.CDATA:
```

```
 Characters characters = event.asCharacters();
 if (characters.isCData()) {
  delegate.writeCData(characters.getData());
 } else {
  delegate.writeCharacters(characters.getData());
 }
 break;
case XMLStreamConstants.COMMENT:
 delegate.writeComment(((Comment) event).getText());
 break;
case XMLStreamConstants.DTD:
 delegate.writeDTD(((DTD) event).getDocumentTypeDeclaration());
 break;
case XMLStreamConstants.ENTITY_REFERENCE:
 delegate.writeEntityRef(((EntityReference) event).getName());
 break;
case XMLStreamConstants.PROCESSING_INSTRUCTION:
 ProcessingInstruction processingInstruction = (ProcessingInstruction) event;
 delegate.writeProcessingInstruction(processingInstruction.getTarget(), processingInstruction.getData());
 break;
case XMLStreamConstants.SPACE:
 break;
default:
 throw new XMLStreamException("Cannot write event " + event);
 }
}

@Override
public void add(XMLEventReader reader) throws XMLStreamException {
 while (reader.peek() != null) {
  add(reader.nextEvent());
 }
}

@Override
public void close() throws XMLStreamException {
 delegate.close();
}

@Override
public void flush() throws XMLStreamException {
 delegate.flush();
}

@Override
public NamespaceContext getNamespaceContext() {
 return delegate.getNamespaceContext();
}
```

```
 @Override
 public String getPrefix(String uri) throws XMLStreamException {
  return delegate.getPrefix(uri);
 }

 @Override
 public void setDefaultNamespace(String uri) throws XMLStreamException {
  delegate.setDefaultNamespace(uri);
 }

 @Override
 public void setNamespaceContext(NamespaceContext ctxt) throws XMLStreamException {
  delegate.setNamespaceContext(ctxt);
 }

 @Override
 public void setPrefix(String prefix, String uri) throws XMLStreamException {
  delegate.setPrefix(prefix, uri);
 }
}
/*
* Copyright 2011, 2012 Odysseus Software GmbH
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package de.odysseus.staxon.event;

import javax.xml.stream.EventFilter;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.EntityReference;
import javax.xml.stream.events.XMLEvent;
import javax.xml.stream.util.EventReaderDelegate;

/**
* Simple implementation of a filtered {@link XMLEventReader}.
```

```
*/
public class SimpleXMLFilteredEventReader extends EventReaderDelegate {
 private final EventFilter filter;

 private int currentEventType = -1;

 public SimpleXMLFilteredEventReader(XMLEventReader reader, EventFilter filter) {
  super(reader);
  this.filter = filter;
 }

 @Override
 public void setParent(XMLEventReader reader) {
  throw new UnsupportedOperationException();
 }

 @Override
 public boolean hasNext() {
  try {
   while (getParent().hasNext()) {
    if (filter.accept(getParent().peek())) {
     return true;
    }
    getParent().nextEvent();
   }
   return false;
  } catch (XMLStreamException e) {
   return false;
  }
 }

 @Override
 public XMLEvent peek() throws XMLStreamException {
  return hasNext() ? getParent().peek() : null;
 }

 @Override
 public Object next() {
  try {
   return nextEvent();
  } catch (XMLStreamException e) {
   throw new RuntimeException(e);
  }
 }

 @Override
 public XMLEvent nextEvent() throws XMLStreamException {
  if (hasNext()) {
```

```
 XMLEvent event = getParent().nextEvent();
 currentEventType = event.getEventType();
 return event;
 }
 throw new XMLStreamException("no more events");
}

@Override
public String getElementText() throws XMLStreamException {
 if (currentEventType != XMLStreamConstants.START_ELEMENT) {
  throw new XMLStreamException("Expected start element event");
 }
 StringBuilder builder = null;
 String leadText = null;
 while (true) {
  XMLEvent event = nextEvent();
  String data = null;
  switch (event.getEventType()) {
  case XMLStreamConstants.ENTITY_REFERENCE:
   data = ((EntityReference) event).getName();
   break;
  case XMLStreamConstants.CDATA:
  case XMLStreamConstants.CHARACTERS:
   data = event.asCharacters().getData();
   break;
  case XMLStreamConstants.COMMENT:
  case XMLStreamConstants.PROCESSING_INSTRUCTION:
  case XMLStreamConstants.SPACE:
   break;
  case XMLStreamConstants.END_ELEMENT:
   return builder == null ? (leadText == null ? "" : leadText) : builder.toString();
  default:
   throw new XMLStreamException("Unexpected event type " + currentEventType, event.getLocation());
  }
  if (data != null) {
   if (leadText == null) { // first event?
    leadText = data;
   } else {
    if (builder == null) { // second event?
     builder = new StringBuilder(leadText);
    }
    builder.append(data);
   }
  }
 }
}

@Override
```

```
 public XMLEvent nextTag() throws XMLStreamException {
  while (true) {
   XMLEvent event = nextEvent();
   switch (event.getEventType()) {
   case XMLStreamConstants.START_DOCUMENT:
    break;
   case XMLStreamConstants.COMMENT:
   case XMLStreamConstants.PROCESSING_INSTRUCTION:
   case XMLStreamConstants.SPACE:
    break;
   case XMLStreamConstants.CDATA:
   case XMLStreamConstants.CHARACTERS:
    if (!event.asCharacters().isWhiteSpace()) {
     throw new XMLStreamException("Encountered non-whitespace text");
    }
    break;
   case XMLStreamConstants.START_ELEMENT:
   case XMLStreamConstants.END_ELEMENT:
    return event;
   default:
    throw new XMLStreamException("Encountered unexpected event: " + event.getEventType());
   }
  }
 }
}
/*
* Copyright 2011, 2012 Odysseus Software GmbH
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package de.odysseus.staxon.json.stream.impl;

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.Reader;
```

```
import java.io.Writer;

import de.odysseus.staxon.json.stream.JsonStreamFactory;
import de.odysseus.staxon.json.stream.JsonStreamSource;
import de.odysseus.staxon.json.stream.JsonStreamTarget;

/**
* Default <code>JsonStreamFactory</code> implementation.
*/
public class JsonStreamFactoryImpl extends JsonStreamFactory {
 private final String prettyIndent;
 private final String prettyNewline;
 private final String prettySpace;

 /**
  * Create instance.
  * Petty printing will use <code>"\t"</code> for indentation (per level),
  * <code>"\n"</code> as line separator and <code>" "</code> to decorate
  * colons, commas, etc.
  */
 public JsonStreamFactoryImpl() {
  this(" ", "\t", "\n");
 }

 /**
  * Create instance.
  * @param prettySpace inserted around colons, commas, etc
  * @param prettyIndent indentation per depth level
  * @param prettyNewline newline character sequence
  */
 public JsonStreamFactoryImpl(String prettySpace, String prettyIndent, String prettyNewline) {
  this.prettySpace = prettySpace;
  this.prettyIndent = prettyIndent;
  this.prettyNewline = prettyNewline;
 }

 @Override
 public JsonStreamSource createJsonStreamSource(InputStream input) throws IOException {
  return createJsonStreamSource(new InputStreamReader(input, "UTF-8"));
 }

 @Override
 public JsonStreamSource createJsonStreamSource(Reader reader) {
  return new JsonStreamSourceImpl(new Yylex(reader), false);
 }

 @Override
 public JsonStreamTarget createJsonStreamTarget(OutputStream output, boolean pretty) throws IOException {
```

```
 return createJsonStreamTarget(new OutputStreamWriter(output, "UTF-8"), pretty);
 }

 @Override
 public JsonStreamTarget createJsonStreamTarget(Writer writer, boolean pretty) {
  if (pretty) {
   return new JsonStreamTargetImpl(writer, false, prettySpace, prettyIndent, prettyNewline);
  } else {
   return new JsonStreamTargetImpl(writer, false);
  }
 }
}
/*
* Copyright 2011, 2012 Odysseus Software GmbH
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package de.odysseus.staxon.json.stream.impl;

import java.io.Closeable;
import java.io.IOException;
import java.math.BigDecimal;
import java.math.BigInteger;

import de.odysseus.staxon.json.stream.JsonStreamSource;
import de.odysseus.staxon.json.stream.JsonStreamToken;

/**
* Default <code>JsonStreamSource</code> implementation.
*/
class JsonStreamSourceImpl implements JsonStreamSource {
 /**
  * Scanner interface
  */
 interface Scanner extends Closeable {
  enum Symbol {
     START_OBJECT,
     END_OBJECT,
```

```
    START_ARRAY,
    END_ARRAY,
    COLON,
    COMMA,
    STRING,
    NUMBER,
    TRUE,
    FALSE,
    NULL,
    EOF;
  }
  Symbol nextSymbol() throws IOException;
  String getText();

  int getCharOffset();
  int getLineNumber();
  int getColumnNumber();
 }

 private final Scanner scanner;
 private final boolean[] arrays = new boolean[64];
 private final boolean closeScanner;

 private JsonStreamToken token = null;
 private Scanner.Symbol symbol = null;
 private int depth = 0;
 private boolean peeked = false;

 private int lineNumber;
 private int columnNumber;
 private int charOffset;


 JsonStreamSourceImpl(Scanner scanner, boolean closeScanner) {
  this.scanner = scanner;
  this.closeScanner = closeScanner;
  this.lineNumber = scanner.getLineNumber();
  this.columnNumber = scanner.getColumnNumber();
  this.charOffset = scanner.getCharOffset();
 }

 private JsonStreamToken startJsonValue() throws IOException {
  switch (symbol) {
  case FALSE:
  case NULL:
  case NUMBER:
  case TRUE:
  case STRING:
```

```
 return JsonStreamToken.VALUE;
case START_ARRAY:
 if (arrays[depth]) {
  throw new IOException("Already in an array");
 }
 arrays[depth] = true;
 return JsonStreamToken.START_ARRAY;
case START_OBJECT:
 depth++;
 return JsonStreamToken.START_OBJECT;
default:
 throw new IOException("Unexpected symbol: " + symbol);
}
}

private void require(Scanner.Symbol expected) throws IOException {
 if (symbol != expected) {
  throw new IOException("Unexpected symbol:" + symbol);
 }
}

private JsonStreamToken next() throws IOException {
 symbol = scanner.nextSymbol();
 if (symbol == Scanner.Symbol.EOF) {
  if (depth != 0 || arrays[depth]) {
   throw new IOException("Premature EOF");
  }
  return JsonStreamToken.NONE;
 }
 if (token == null) {
  return startJsonValue();
 }
 switch (token) {
 case NAME:
  require(Scanner.Symbol.COLON);
  symbol = scanner.nextSymbol();
  return startJsonValue();
 case END_OBJECT:
 case END_ARRAY:
 case VALUE:
  switch (symbol) {
  case COMMA:
   symbol = scanner.nextSymbol();
   if (arrays[depth]) {
    return startJsonValue();
   } else {
    require(Scanner.Symbol.STRING);
    return JsonStreamToken.NAME;
```

```
  }
 case END_ARRAY:
  if (!arrays[depth]) {
   throw new IOException("Not in an array");
  }
  arrays[depth] = false;
  return JsonStreamToken.END_ARRAY;
 case END_OBJECT:
  if (arrays[depth]) {
   throw new IOException("Unclosed array");
  }
  if (depth == 0) {
   throw new IOException("Not in an object");
  }
  depth--;
  return JsonStreamToken.END_OBJECT;
 default:
  throw new IOException("Unexpected symbol: " + symbol);
 }
case START_OBJECT:
 switch (symbol) {
 case END_OBJECT:
  depth--;
  return JsonStreamToken.END_OBJECT;
 case STRING:
  return JsonStreamToken.NAME;
 default:
  throw new IOException("Unexpected symbol: " + symbol);
 }
case START_ARRAY:
 switch (symbol) {
 case END_ARRAY:
  arrays[depth] = false;
  return JsonStreamToken.END_ARRAY;
 default:
  return startJsonValue();
 }
default:
 throw new IOException("Unexpected token: " + token);
}
}

@Override
public void close() throws IOException {
 if (closeScanner) {
  scanner.close();
 }
}
```

```
/**
 * Make the next token the current token.
 * Save location info from scanner to prevent changing location by peek()
 * @param token expected token
 * @throws IOException
 */
private void poll(JsonStreamToken token) throws IOException {
 if (token != peek()) {
  throw new IOException("Unexpected token: " + peek());
 }
 lineNumber = scanner.getLineNumber();
 columnNumber = scanner.getColumnNumber();
 charOffset = scanner.getCharOffset();
 peeked = false;
}

@Override
public String name() throws IOException {
 poll(JsonStreamToken.NAME);
 return scanner.getText();
}

@Override
public Value value() throws IOException {
 poll(JsonStreamToken.VALUE);
 switch (symbol) {
 case NULL:
  return NULL;
 case STRING:
  return new Value(scanner.getText());
 case TRUE:
  return TRUE;
 case FALSE:
  return FALSE;
 case NUMBER:
  if (scanner.getText().indexOf('.') < 0 && scanner.getText().toLowerCase().indexOf('e') < 0) {
   return new Value(scanner.getText(), new BigInteger(scanner.getText()));
  } else {
   return new Value(scanner.getText(), new BigDecimal(scanner.getText()));
  }
 default:
  throw new IOException("Not a value token: " + symbol);
 }
}

@Override
public void startObject() throws IOException {
```

```
 poll(JsonStreamToken.START_OBJECT);
}

@Override
public void endObject() throws IOException {
 poll(JsonStreamToken.END_OBJECT);
}

@Override
public void startArray() throws IOException {
 poll(JsonStreamToken.START_ARRAY);
}

@Override
public void endArray() throws IOException {
 poll(JsonStreamToken.END_ARRAY);
}

@Override
public JsonStreamToken peek() throws IOException {
 if (!peeked) {
  token = next();
  peeked = true;
 }
 return token;
}

@Override
public int getLineNumber() {
 return lineNumber + 1;
}

@Override
public int getColumnNumber() {
 return columnNumber + 1;
}

@Override
public int getCharacterOffset() {
 return charOffset;
}

@Override
public String getPublicId() {
 return null;
}

@Override
```

```
 public String getSystemId() {
  return null;
 }
}
/*
* Copyright 2011, 2012 Odysseus Software GmbH
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package de.odysseus.staxon.json.stream.impl;

import java.io.IOException;
import java.io.Writer;

import de.odysseus.staxon.json.stream.JsonStreamTarget;

/**
* Default <code>JsonStreamTarget</code> implementation.
*/
class JsonStreamTargetImpl implements JsonStreamTarget {
 private final Writer writer;
 private final int[] namePos = new int[64];
 private final int[] arrayPos = new int[64];
 private final StringBuilder buffer = new StringBuilder();
 private final boolean closeWriter;

 private final String[] indent;
 private final String space;

 private int depth = 0;

 JsonStreamTargetImpl(Writer writer, boolean closeWriter) {
  this(writer, closeWriter, null, null, null);
 }

 JsonStreamTargetImpl(Writer writer, boolean closeWriter, String prettySpace, String prettyIndent, String
prettyNewline) {
  this.writer = writer;
```

```
 this.closeWriter = closeWriter;
 this.space = prettySpace;

 if (prettyIndent != null || prettyNewline != null) {
  this.indent = new String[64];
  StringBuilder builder = new StringBuilder();
  if (prettyNewline != null) {
   builder.append(prettyNewline);
  }
  for (int i = 0; i < 64; i++) {
   indent[i] = builder.toString();
   if (prettyIndent != null) {
    builder.append(prettyIndent);
   }
  }
 } else {
  this.indent = null;
 }
}

private String encode(String value) {
 buffer.setLength(0);
 for (int i = 0; i < value.length(); i++) {
  char c = value.charAt(i);
  switch (c) {
  case '"':
   buffer.append("\\\"");
   break;
  case '\\':
   buffer.append("\\\\");
   break;
  case '\b':
   buffer.append("\\b");
   break;
  case '\f':
   buffer.append("\\f");
   break;
  case '\n':
   buffer.append("\\n");
   break;
  case '\r':
   buffer.append("\\r");
   break;
  case '\t':
   buffer.append("\\t");
   break;
  default:
   if (c < ' ') {
```

```
    buffer.append(String.format("\\u%04X", (int) c));
   } else {
    buffer.append(c);
   }
  }
 }
 return buffer.toString();
}

@Override
public void close() throws IOException {
 if (closeWriter) {
  writer.close();
 } else {
  writer.flush();
 }
}

@Override
public void flush() throws IOException {
 writer.flush();
}

@Override
public void name(String name) throws IOException {
 if (namePos[depth] > 1) {
  writer.write(',');
 }
 namePos[depth]++;
 if (indent != null) {
  writer.write(indent[depth]);
 } else if (space != null) {
  writer.write(space);
 }
 writer.write("");
 writer.write(name);
 writer.write("");
 if (space != null) {
  writer.write(space);
 }
 writer.write(':');
}

@Override
public void value(Object value) throws IOException {
 if (arrayPos[depth] > 0) {
  if (arrayPos[depth] > 1) {
   writer.write(',');
```

```
 }
 arrayPos[depth]++;
 }
 if (space != null) {
  writer.write(space);
 }
 if (value == null) {
  writer.write("null");
 } else if (value instanceof String) {
  writer.write("");
  writer.write(encode((String) value));
  writer.write("");
 } else {
  writer.write(value.toString());
 }
}

@Override
public void startObject() throws IOException {
 if (arrayPos[depth] > 0) {
  if (arrayPos[depth] > 1) {
   writer.write(',');
  }
  arrayPos[depth]++;
 }
 if (space != null && (depth > 0 || arrayPos[depth] > 0)) {
  writer.write(space);
 }
 writer.write('{');
 depth++;
 namePos[depth] = 1;
}

@Override
public void endObject() throws IOException {
 namePos[depth] = 0;
 depth--;
 if (indent != null) {
  writer.write(indent[depth]);
 } else if (space != null) {
  writer.write(space);
 }
 writer.write('}');
 if (depth == 0) {
  writer.flush();
 }
}
```

```
 @Override
 public void startArray() throws IOException {
  if (arrayPos[depth] > 0) {
   throw new IOException("Nested arrays are not supported!");
  }
  if (space != null && depth > 0) {
   writer.write(space);
  }
  writer.write('[');
  arrayPos[depth] = 1;
 }

 @Override
 public void endArray() throws IOException {
  arrayPos[depth] = 0;
  if (space != null) {
   writer.write(space);
  }
  writer.write(']');
 }
}
/*
* Copyright 2011, 2012 Odysseus Software GmbH
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package de.odysseus.staxon.json;

import java.util.Map;

import javax.xml.namespace.QName;

/**
* <p>Simple JSON XML configuration.</p>
*
* <p>Initially, values are set according to {@link JsonXMLConfig#DEFAULT}.</p>
* @see JsonXMLConfig
*/
```

```
public class JsonXMLConfigImpl implements JsonXMLConfig, Cloneable {
 private QName virtualRoot = JsonXMLConfig.DEFAULT.getVirtualRoot();
 private boolean multiplePI = JsonXMLConfig.DEFAULT.isMultiplePI();
 private boolean prettyPrint = JsonXMLConfig.DEFAULT.isPrettyPrint();
 private boolean autoArray = JsonXMLConfig.DEFAULT.isAutoArray();
 private boolean autoPrimitive = JsonXMLConfig.DEFAULT.isAutoPrimitive();
 private boolean namespaceDeclarations = JsonXMLConfig.DEFAULT.isNamespaceDeclarations();
 private char namespaceSeparator = JsonXMLConfig.DEFAULT.getNamespaceSeparator();
 private Map<String, String> namespaceMappings = JsonXMLConfig.DEFAULT.getNamespaceMappings();

 private boolean repairingNamespaces = JsonXMLConfig.DEFAULT.isRepairingNamespaces();

 @Override
 protected JsonXMLConfigImpl clone() {
  try {
   return (JsonXMLConfigImpl) super.clone();
  } catch (CloneNotSupportedException e) {
   throw new RuntimeException(e); // should not happen
  }
 }

 @Override
 public boolean isAutoArray() {
  return autoArray;
 }

 public void setAutoArray(boolean autoArray) {
  this.autoArray = autoArray;
 }

 @Override
 public boolean isAutoPrimitive() {
  return autoPrimitive;
 }

 public void setAutoPrimitive(boolean autoPrimitive) {
  this.autoPrimitive = autoPrimitive;
 }

 @Override
 public boolean isMultiplePI() {
  return multiplePI;
 }

 public void setMultiplePI(boolean multiplePI) {
  this.multiplePI = multiplePI;
 }
```

```
@Override
public boolean isNamespaceDeclarations() {
 return namespaceDeclarations;
}

public void setNamespaceDeclarations(boolean namespaceDeclarations) {
 this.namespaceDeclarations = namespaceDeclarations;
}

@Override
public char getNamespaceSeparator() {
 return namespaceSeparator;
}

public void setNamespaceSeparator(char namespaceSeparator) {
 this.namespaceSeparator = namespaceSeparator;
}

@Override
public boolean isPrettyPrint() {
 return prettyPrint;
}

public void setPrettyPrint(boolean prettyPrint) {
 this.prettyPrint = prettyPrint;
}

@Override
public QName getVirtualRoot() {
 return virtualRoot;
}

public void setVirtualRoot(QName virtualRoot) {
 this.virtualRoot = virtualRoot;
}

@Override
public boolean isRepairingNamespaces() {
 return repairingNamespaces;
}

public void setRepairingNamespaces(boolean repairingNamespaces) {
 this.repairingNamespaces = repairingNamespaces;
}

@Override
public Map<String, String> getNamespaceMappings() {
 return namespaceMappings;
```

```
 }

 public void setNamespaceMappings(Map<String, String> namespaceMappings) {
  this.namespaceMappings = namespaceMappings;
 }
}
```