JSR-000206 Java™API for XML Processing ("Specification")

Version: 1.6
December 4, 2013

Maintenance Lead:
   Joe Wang
   Oracle Corporation

Feedback:
   Please send comments to eg@jaxp.java.net

Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA
ORACLE AMERICA, INC. IS WILLING TO LICENSE THIS SPECIFICATION TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS AGREEMENT. PLEASE READ THE TERMS AND CONDITIONS OF THIS AGREEMENT CAREFULLY. BY DOWNLOADING THIS SPECIFICATION, YOU ACCEPT THE TERMS AND CONDITIONS OF THE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY IT, SELECT THE "DECLINE" BUTTON AT THE BOTTOM OF THIS PAGE.

Specification: JSR-206 Java™API for XML Processing ("Specification")

Version: 1.6

Status: Final Release

Specification Lead: Oracle America, Inc. ("Specification Lead")

Release: October 2013

Copyright 2013 Oracle America, Inc.
All rights reserved.

LIMITED LICENSE GRANTS

1. License for Evaluation Purposes. Specification Lead hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense), under Specification Lead's applicable intellectual property rights to view, download, use and reproduce the Specification only for the purpose of internal evaluation. This includes (i) developing applications intended to run on an implementation of the Specification, provided that such applications do not themselves implement any portion(s) of the Specification, and (ii) discussing the Specification with any third party; and (iii) excerpting brief portions of the Specification in oral or written communications which discuss the Specification provided that such excerpts do not in the aggregate constitute a significant portion of the Specification.

2. License for the Distribution of Compliant Implementations. Specification Lead also grants you a perpetual, non-exclusive, non-transferable, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights or, subject to the provisions of subsection 4 below, patent rights it may have covering the Specification to create and/or distribute an Independent Implementation of the Specification that: (a) fully implements the Specification including all its required interfaces and functionality; (b) does not modify, subset, superset or otherwise extend the Licensor Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the Licensor Name Space other than those required/authorized by the Specification or Specifications being implemented; and (c) passes the Technology Compatibility Kit (including satisfying the requirements of the applicable TCK Users Guide) for such Specification ("Compliant Implementation"). In addition, the foregoing license is expressly conditioned on your not acting outside its scope. No license is granted hereunder for any other purpose (including, for example, modifying the Specification, other than to the extent of your fair use rights, or distributing the Specification to third parties). Also, no right, title, or interest in or to any trademarks, service marks, or trade names of Specification Lead or Specification Lead's licensors is granted hereunder. Java, and Java-related logos, marks and names are trademarks or registered trademarks of Oracle America, Inc. in the U.S. and other countries.
3. Pass-through Conditions. You need not include limitations (a)-(c) from the previous paragraph or any other particular "pass through" requirements in any license You grant concerning the use of your Independent Implementation or products derived from it. However, except with respect to Independent Implementations (and products derived from them) that satisfy limitations (a)-(c) from the previous paragraph, You may neither: (a) grant or otherwise pass through to your licensees any licenses under Specification Lead's applicable intellectual property rights; nor (b) authorize your licensees to make any claims concerning their implementation's compliance with the Specification in question.

   a. With respect to any patent claims covered by the license granted under subparagraph 2 above that would be infringed by all technically feasible implementations of the Specification, such license is conditioned upon your offering on fair, reasonable and non-discriminatory terms, to any party seeking it from You, a perpetual, non-exclusive, non-transferable, worldwide license under Your patent rights which are or would be infringed by all technically feasible implementations of the Specification to develop, distribute and use a Compliant Implementation.

   b. With respect to any patent claims owned by Specification Lead and covered by the license granted under subparagraph 2, whether or not their infringement can be avoided in a technically feasible manner when implementing the Specification, such license shall terminate with respect to such claims if You initiate a claim against Specification Lead that it has, in the course of performing its responsibilities as the Specification Lead, induced any other entity to infringe Your patent rights.

   c. Also with respect to any patent claims owned by Specification Lead and covered by the license granted under subparagraph 2 above, where the infringement of such claims can be avoided in a technically feasible manner when implementing the Specification such license, with respect to such claims, shall terminate if You initiate a claim against Specification Lead that its making, having made, using, offering to sell, selling or importing a Compliant Implementation infringes Your patent rights.

5. Definitions. For the purposes of this Agreement: "Independent Implementation" shall mean an implementation of the Specification that neither derives from any of Specification Lead's source code or binary code materials nor, except with an appropriate and separate license from Specification Lead, includes any of Specification Lead's source code or binary code materials; "Licensor Name Space" shall mean the public class or interface declarations whose names begin with "java", "javax", "com.oracle", "com.sun" or their equivalents in any subsequent naming convention adopted by Oracle America, Inc. through the Java Community Process, or any recognized successors or replacements thereof; and "Technology Compatibility Kit" or "TCK" shall mean the test suite and accompanying TCK User's Guide provided by Specification Lead which corresponds to the Specification and that was available either (i) from Specification Lead's 120 days before the first release of Your Independent Implementation that allows its use for commercial purposes, or (ii) more recently than 120 days from such release but against which You elect to test Your implementation of the Specification.

This Agreement will terminate immediately without notice from Specification Lead if you breach the Agreement or act outside the scope of the licenses granted above.
DISCLAIMER OF WARRANTIES

THE SPECIFICATION IS PROVIDED "AS IS". SPECIFICATION LEAD MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT (INCLUDING AS A CONSEQUENCE OF ANY PRACTICE OR IMPLEMENTATION OF THE SPECIFICATION), OR THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE. This document does not represent any commitment to release or implement any portion of the Specification in any product. In addition, the Specification could include technical inaccuracies or typographical errors.

LIMITATION OF LIABILITY

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SPECIFICATION LEAD OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED IN ANY WAY TO YOUR HAVING, IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION, EVEN IF SPECIFICATION LEAD AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend Specification Lead and its licensors from any claims arising or resulting from: (i) your use of the Specification; (ii) the use or distribution of your Java application, applet and/or implementation; and/or (iii) any claims that later versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

REstricted RIGHTS LEGEND

U.S. Government: If this Specification is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

REPORT

If you provide Specification Lead with any comments or suggestions concerning the Specification ("Feedback"), you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Specification Lead a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose.

GENERAL TERMS

Any action related to this Agreement will be governed by California law and controlling U.S. federal law. The U.N. Convention for the International Sale of Goods and the choice of law rules of any jurisdiction will not apply.

The Specification is subject to U.S. export control laws and may be subject to export or import regulations in other countries. Licensee agrees to comply strictly with all such laws and regulations and acknowledges that it has the responsibility to obtain such licenses to export, re-export or import as may be required after delivery to
Licensee.

This Agreement is the parties' entire agreement relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, conditions, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification to this Agreement will be binding, unless in writing and signed by an authorized representative of each party.
# Table of Contents

1. Overview.........................................................................................................................8
   1.1 What is XML?................................................................................................................8
   1.2 XML and the Java™ Platform........................................................................................8
   1.3 About This Specification...............................................................................................8
   1.4 Who Should Read This Document................................................................................8
   1.5 Report and Contact........................................................................................................9
   1.6 Development of This Specification..............................................................................9
   1.7 Development of the JAXP 1.3 Specification...............................................................9
   1.8 Development of the JAXP 1.4 Specification...............................................................9
   1.9 Development of the JAXP 1.5 Specification................................................................10
   1.10 Development of the JAXP 1.6 Specification.............................................................10
   1.11 Acknowledgments........................................................................................................10

2. Related Technologies.........................................................................................................11
   2.1 Streaming API for XML (StAX)..................................................................................11
   2.2 XML 1.1 Support in StAX...........................................................................................11

3. Endorsed Specifications.....................................................................................................12
   3.1 Extensible Markup Language (XML).........................................................................12
   3.2 Namespaces in XML......................................................................................................12
   3.3 XML Schema................................................................................................................12
   3.4 XSL Transformations (XSLT).....................................................................................13
   3.5 XML Path Language (XPath).....................................................................................13
   3.6 XML Inclusions (XInclude).........................................................................................13
   3.7 Document Object Model (DOM) Level 2....................................................................13
   3.8 Document Object Model (DOM) Level 3....................................................................14
   3.9 Simple API for XML (SAX).........................................................................................14

4. Plugability Layer..................................................................................................................15
   4.1 SAX Plugability................................................................................................................15
   4.2 Examples........................................................................................................................16
   4.3 DOM Plugability.............................................................................................................16
   4.4 Reliance on SAX API....................................................................................................17
   4.5 Examples........................................................................................................................17
   4.6 XSLT Plugability.............................................................................................................18
   4.7 Examples........................................................................................................................18
   4.8 XPath Plugability............................................................................................................23
   4.9 Examples........................................................................................................................24
   4.10 Validation Plugability....................................................................................................24
   4.11 Streaming API for XML Plugability.............................................................................25
   4.12 Examples........................................................................................................................25
   4.13 Datatype Plugability.......................................................................................................26
   4.14 Thread Safety................................................................................................................27
   4.15 Properties For Enabling Schema Validation.............................................................27
   4.16 Samples Using the Properties....................................................................................29
   4.17 Recommended Implementation of Properties.............................................................30

5. Conformance Requirements...............................................................................................30
1. Overview

1.1 What is XML?

XML is the meta language defined by the World Wide Web Consortium (W3C) that can be used to describe a broad range of hierarchical mark up languages. It is a set of rules, guidelines, and conventions for describing structured data in a plain text, editable file. Using a text format instead of a binary format allows the programmer or even an end user to look at or utilize the data without relying on the program that produced it. However the primary producer and consumer of XML data is the computer program and not the end user. Like HTML, XML makes use of tags and attributes. Tags are words bracketed by the “<” and “>” characters and attributes are strings of the form 'name="value"' that are inside of tags. While HTML specifies what each tag and attribute means, as well as their presentation attributes in a browser, XML uses tags only to delimit pieces of data and leaves the interpretation of the data to the application that uses it. In other words, XML defines only the structure of the document and does not define any of the presentation semantics of that document.

Development of XML started in 1996 leading to a W3C Recommendation in February of 1998. However, the technology is not entirely new. It is based on SGML (Standard Generalized Markup Language) which was developed in the early 1980's and became an ISO standard in 1986. SGML has been widely used for large documentation projects and there is a large community that has experience working with SGML. The designers of XML took the best parts of SGML, used their experience as a guide and produced a technology that is just as powerful as SGML, but much simpler and easier to use. XML-based documents can be used in a wide variety of applications including vertical markets, e-commerce, business-to-business communication, and enterprise application messaging.

1.2 XML and the Java™ Platform

In many ways, XML and the Java Platform form an ideal partnership. XML defines a cross platform data format and Java provides a standard cross platform programming platform. Together, XML and Java technologies allow programmers to apply Write Once, Run Anywhere™ fundamentals to the processing of data and documents generated by both Java based programs and non-Java based programs.

1.3 About This Specification

This document describes the Java API for XML Processing, Version 1.4. This version of the specification introduces basic support for parsing and manipulating XML documents through a standardized set of Java Platform APIs.

When this specification is final there will be a Reference Implementation which will demonstrate the capabilities of this API and will provide an operational definition of the specification. A Technology Compatibility Kit (TCK) will also be available that will verify whether an implementation of this specification is compliant. These are required as per the Java Community Process 2.5 (JCP 2.5).

1.4 Who Should Read This Document

This specification is intended for use by:

- Parser Developers wishing to implement this version of the specification in their parser.
- Application Developers who use the APIs described in this specification and wish to have a more complete understanding of the API.
- Implementors of XPath and XSLT who wish to support the APIs in this specification, particularly the
Transformation and XPath APIs.

This specification is not a tutorial or a user's guide to XML, DOM, SAX, StAX or XSLT. Familiarity with these technologies and specifications on the part of the reader is assumed.

1.5 Report and Contact

Your comments on this specification are welcome and appreciated. Without your comments, the specifications developed under the auspices of the Java Community Process would not serve your needs as well. To comment on this specification, please send email to <eg@jaxp.dev.java.net>.

You can stay current with Sun's Java Platform related activities, as well as information on our <xml-interest> and <xml-announce> mailing lists, at our website [http://java.sun.com/xml/jaxp].

1.6 Development of This Specification

This is a maintenance release of JAXP 1.4.

1.7 Development of the JAXP 1.3 Specification

The JAXP 1.3 specification was developed in accordance with the Java Community Process [http://www.jcp.org] 2.5. It was developed under the authorization of Java Specification Request 206 [http://jcp.org/en/jsr/detail?id=206].

The expert group who contributed to JAXP 1.3 was composed of individuals from a number of companies:

• Ben Galbraith
• Neil Graham, IBM
• Phil Hanna, SAS Institute Inc.
• Todd Karakashian, BEA
• K. Karun, Oracle
• Kohsuke Kawaguchi, Sun Microsystems, Inc.
• Dongeun Kim, Tmax Soft, Inc.
• Harish Krishnaswamy, Novell, Inc.
• Miles Sabin
• Vladimir Savchenko, SAP AG
• Ilene Seelemann, IBM
• Jeff Suttor, Sun Microsystems, Inc.
• Norman Walsh (Specification Lead), Sun Microsystems, Inc.
• Henry Zongaro, IBM

We would like to acknowledge that a lot of the grammar caching work introduced in this version of the specification was derived from the work being done under the Xerces project at Apache.

1.8 Development of the JAXP 1.4 Specification

The informal expert group who contributed to this maintenance release is composed of individuals from a number of companies. These individuals are:

• Neeraj Bajaj, Sun Microsystems, Inc.
• Ben Galbraith
• Michael Glavassevich, IBM
1.9 Development of the JAXP 1.5 Specification

The informal expert group who contributed to this maintenance release is composed of the following individuals:

- Michael Glavassevich, IBM
- Alan Bateman, Oracle Corporation
- Lance Andersen, Oracle Corporation
- Andrew Gross, Oracle Corporation
- Daniel Fuchs, Oracle Corporation
- Joe Wang, Oracle Corporation

1.10 Development of the JAXP 1.6 Specification

The informal expert group who contributed to this maintenance release is composed of the following individuals:

- Michael Glavassevich, IBM
- Alan Bateman, Oracle Corporation
- Lance Andersen, Oracle Corporation
- Daniel Fuchs, Oracle Corporation
- Joe Wang, Oracle Corporation

1.11 Acknowledgments

Many individuals and companies have given their time and talents to the specifications that this specification relies upon. The authors of this specification would like to thank (in no particular order):

- David Brownell, David Megginson and the XML-DEV community who developed the SAX API
- The W3C DOM Working Group chaired by Philippe Le Hégaret
- The Apache Software Foundation [http://apache.org/], for Xerces and Xalan
- Michael Kay, Saxonica [http://www.saxonica.com/]
- Han Ming Ong, Apple
- Eduardo Pelegri-Lopart, Tom Kincaid, Connie Weiss, Gopal Sharma, Neeraj Bajaj, K. Venugopal, Arun
- Yadav, Ramesh Mandava, Bhakti Mehta, Prasad Subramanian, Todd Miller, Joesph Fialli, and Rajiv
- Mordani all of whom work at Sun Microsystems, Inc. and whose talents have all reflected upon the development of this API.
- Margaret L. Wade for allowing it all to be true.
2. Related Technologies

This specification builds upon other technologies developed within the Java Community Process [http://jcp.org/]. Each technology used by this document is called out together with the exact version of the specification and its publicly accessible location.

2.1 Streaming API for XML (StAX)

This specification supports Streaming API for XML (StAX) [http://jcp.org/en/jsr/detail?id=173]. This specification includes by reference Streaming API for XML (StAX) Version 1.0 (JSR 173) in its entirety for the purposes of defining Streaming API for XML (StAX) in the APIs defined herein. The API packages included by reference are:

- javax.xml.stream
- javax.xml.stream.events
- javax.xml.stream.util

2.2 XML 1.1 Support in StAX

The XML 1.1 Recommendation was published after the StAX APIs. As a result, the StAX APIs make no reference to XML 1.1. However, the JAXP API, since version 1.3, has supported XML 1.1. In order to assure the widest possible interoperability and the least amount of user confusion, JAXP 1.4 imposes the additional requirement that the StAX APIs used in a JAXP context must support XML 1.1. We hope and expect that a future version of StAX will officially endorse XML 1.1 and make this additional requirement redundant.
3. Endorsed Specifications

This specification endorses and builds upon several external specifications. Each specification endorsed by this document is called out together with the exact version of the specification and its publicly accessible location. All of these standards have conformance tests provided in the Technology Compatibility Kit available for this specification.

3.1 Extensible Markup Language (XML)


XML is a product of the W3C XML Activity [http://www.w3.org/XML/].

This specification includes by reference Extensible Markup Language (XML) 1.1, XML 1.1 First Edition Specification Errata, Extensible Markup Language (XML) 1.0 (Third Edition) and Extensible Markup Language (XML) 1.0 (Third Edition) Errata in their entirety for the purposes of defining XML in the APIs defined herein.

A Note about XML Versions

XML 1.0 and XML 1.1 are not completely interchangeable. Developers working in environments where a mixture of versions may exist must consider carefully how serialization and transformation may interact with XML versions.

3.2 Namespaces in XML

This specification supports Namespaces in XML 1.1 [http://www.w3.org/TR/xml-names11/] (and Namespaces in XML 1.1 Errata [http://www.w3.org/XML/2004/xml-names11-errata]) and Namespaces in XML 1.0 [http://www.w3.org/TR/REC-xml-names] (and Namespaces in XML 1.0 Errata [http://www.w3.org/XML/xml-names-19990114-errata]).

Namespaces in XML is a product of the W3C XML Activity [http://www.w3.org/XML/].

This specification includes by reference Namespaces in XML 1.1, Namespaces in XML 1.1 Errata, Namespaces in XML 1.0 and Namespaces in XML 1.0 Errata, in their entirety for the purposes of defining Namespaces in XML in the APIs defined herein.

3.3 XML Schema


3.4 XSL Transformations (XSLT)

This specification supports XSL Transformations (XSLT) Version 1.0 [http://www.w3.org/TR/xslt] and XSL Transformations (XSLT) Version 1.0 Errata [http://www.w3.org/1999/11/REC-xslt-19991116-errata].

XSLT is a product of the W3C Style Activity [http://www.w3.org/Style/Activity].

This specification includes by reference XSL Transformations (XSLT) Version 1.0 in its entirety for the purposes of defining XSLT in the APIs defined herein.

3.5 XML Path Language (Xpath)

This specification supports XML Path Language (XPath) Version 1.0 [http://www.w3.org/TR/xpath] and XML Path Language (XPath) Version 1.0 Errata [http://www.w3.org/1999/11/REC-xpath-19991116-errata].

XPath is a product of the W3C XML Activity [http://www.w3.org/XML/Activity] and W3C Style Activity [http://www.w3.org/Style/Activity].

This specification includes by reference XML Path Language (XPath) Version 1.0 and XML Path Language (XPath) Version 1.0 Errata in their entirety for the purposes of defining SAX in the APIs defined herein.

3.6 XML Inclusions (XInclude)

This specification supports XML Inclusions (XInclude) Version 1.0 [http://www.w3.org/TR/xinclude/].

XInclude is a product of the W3C XML Core Working Group [http://www.w3.org/XML/Core/] as part of the W3C XML Activity [http://www.w3.org/XML/Activity].

This specification includes by reference XML Inclusions (XInclude) Version 1.0, in its entirety for the purposes of defining XInclude in the APIs defined herein.

3.7 Document Object Model (DOM) Level 2


DOM Level 2 is a product of the W3C DOM Activity [http://www.w3.org/DOM/].

This specification includes by reference Document Object Model (DOM) Level 2 Core, Document Object Model (DOM) Level 2 Traversal and Range, and Document Object Model (DOM) Level 2 Events in their entirety for the purposes of defining DOM in the APIs defined herein.

The API packages included by reference are:

• org.w3c.dom
3.8 Document Object Model (DOM) Level 3

This specification supports Document Object Model (DOM) Level 3 Core [http://www.w3.org/TR/DOM-Level-3-Core] and Document Object Model (DOM) Level 3 Load and Save [http://www.w3.org/TR/DOM-Level-3-LS].

DOM Level 3 is a product of the W3C DOM Activity [http://www.w3.org/DOM/].

This specification includes by reference Document Object Model (DOM) Level 3 Core and Document Object Model (DOM) Level 3 Load and Save in their entirety for the purposes of defining DOM in the APIs defined herein.

The API packages included by reference are:
• org.w3c.dom
• org.w3c.dom.bootstrap
• org.w3c.dom.events
• org.w3c.dom.ls

3.9 Simple API for XML (SAX)

This specification supports Simple API for XML (SAX) 2.0.2 (sax2r3) [http://sax.sourceforge.net/] and Simple API for XML (SAX) 2.0.2 (sax2r3) Extensions [http://sax.sourceforge.net/?selected=ext].

Simple API for XML (SAX) 2.0.2 (sax2r3) is a product of the SAX Community [http://sax.sourceforge.net/].

This specification includes by reference Simple API for XML (SAX) 2.0.2 (sax2r3) and Simple API for XML (SAX) 2.0.2 (sax2r3) Extensions in their entirety for the purposes of defining Simple API for XML (SAX) 2.0.2 (sax2r3) in the APIs defined herein.

The API packages included by reference are:
• org.xml.sax
• org.xml.sax.ext
• org.xml.sax.helpers
4. Plugability Layer

The endorsed APIs provide broad and useful functionality. However, using the endorsed APIs often requires knowledge of the specific implementations available. Providing the functionality of the endorsed APIs in the Java Platform, while allowing choice of the implementation of the parser, requires a Plugability layer.

This section of the specification defines a Plugability mechanism to allow any compliant implementations to be used through the APIs defined in this specification.

4.1 SAX Plugability

The SAX Plugability classes allow an application programmer to provide an implementation of the org.xml.sax.DefaultHandler API to a SAXParser implementation and parse XML documents. As the parser processes the XML document, it will call methods on the provided DefaultHandler.

In order to obtain a SAXParser instance, an application programmer first obtains an instance of a SAXParserFactory. The SAXParserFactory instance is obtained via one of the static newInstance methods of the SAXParserFactory class.

This method uses the following ordered lookup procedure to determine the SAXParserFactory implementation class to load:

- Use the javax.xml.parsers.SAXParserFactory system property.

- Use the properties file "lib/jaxp.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the system property defined above. The jaxp.properties file is read only once by the JSR-000206 Java™API for XML Processing (“Specification”) implementation and its values are then cached for future use. If the file does not exist when the first attempt is made to read from it, no further attempts are made to check for its existence. It is not possible to change the value of any property in jaxp.properties after it has been read for the first time.

- Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for the classname in the file META-INF/services(javax.xml.parsers.SAXParserFactory) in jars available to the runtime.

- Platform default SAXParserFactory instance.

If the SAXParserFactory implementation class cannot be loaded or instantiated at runtime, a FactoryConfigurationException is thrown. This error message should contain a descriptive explanation of the problem and how the user can resolve it.

The instance of SAXParserFactory can optionally be configured by the application programmer to provide parsers that are namespace aware, or validating, or both. These settings are made using the setNamespaceAware and setValidating methods of the factory. The application programmer can then obtain a SAXParser implementation instance from the factory. If the factory cannot provide a parser configured as set by the application programmer, then a ParserConfigurationException is thrown.
4.2 Examples

The following is a simple example of how to parse XML content from a URL:

```java
SAXParser parser;
DefaultHandler handler = new MyApplicationParseHandler();
SAXParserFactory factory = SAXParserFactory.newInstance();
try {
    parser = factory.newSAXParser();
    parser.parse("http://myserver/mycontent.xml", handler);
} catch (SAXException se) {
    // handle error
} catch (IOException ioe) {
    // handle error
} catch (ParserConfigurationException pce) {
    // handle error
}
```

The following is an example of how to configure a SAX parser to be namespace aware and validating:

```java
SAXParser parser;
DefaultHandler handler = new MyApplicationParseHandler();
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setNamespaceAware(true);
factory.setValidating(true);
try {
    parser = factory.newSAXParser();
    parser.parse("http://myserver/mycontent.xml", handler);
} catch (SAXException se) {
    // handle error
} catch (IOException ioe) {
    // handle error
} catch (ParserConfigurationException pce) {
    // handle error
}
```

An example of how one could pass the System property as a command line option is:

```
java -Djavax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
user.parserApp
```

4.3 DOM Plugability

The DOM plugability classes allow a programmer to parse an XML document and obtain an `org.w3c.dom.Document` object from a `DocumentBuilder` implementation which wraps an underlying DOM implementation.

In order to obtain a `DocumentBuilder` instance, an application programmer first obtains an instance of a `DocumentBuilderFactory`. The `DocumentBuilderFactory` instance is obtained via one of the static `newInstance` methods of the `DocumentBuilderFactory` class.

This method uses the following ordered lookup procedure to determine the `DocumentBuilderFactory` implementation class to load:

- Use the `javax.xml.parsers.DocumentBuilderFactory` system property
• Use the properties file "lib/jaxp.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the system property defined above. The jaxp.properties file is read only once by the JSR-000206 Java™API for XML Processing (“Specification”) implementation and its values are then cached for future use. If the file does not exist when the first attempt is made to read from it, no further attempts are made to check for its existence. It is not possible to change the value of any property in jaxp.properties after it has been read for the first time.

• Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for the classname in the file META-INF/servicesjavax.xml.parsers.DocumentBuilderFactory in jars available to the runtime.

• Platform default DocumentBuilderFactory instance.

If the DocumentBuilderFactory implementation class cannot be loaded or instantiated at runtime, a FactoryConfigurationException is thrown. This error message should contain a descriptive explanation of the problem and how the user can resolve it.

The instance of DocumentBuilderFactory can optionally be configured by the application programmer to provide parsers that are namespace aware or validating, or both. These settings are made using the setNamespaceAware and setValidating methods of the factory. The application programmer can then obtain a DocumentBuilder implementation instance from the factory. If the factory cannot provide a parser configured as set by the application programmer, then a ParserConfigurationException is thrown.

4.4 Reliance on SAX API

The DocumentBuilder reuses several classes from the SAX API. This does not mean that the implementor of the underlying DOM implementation must use a SAX parser to parse the XML content, only that the implementation communicate with the application using these existing and defined APIs.

4.5 Examples

The following is a simple example of how to parse XML content from a URL:

```java
DocumentBuilder builder;
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
String location = "http://myserver/mycontent.xml";
try {
    builder = factory.newDocumentBuilder();
    Document document = builder.parse(location);
} catch (SAXException se) {
    // handle error
} catch (IOException ioe) {
    // handle error
} catch (ParserConfigurationException pce) {
    // handle error
}
```

The following is an example of how to configure a factory to produce parsers to be namespace aware and validating:

```java
DocumentBuilder builder;
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```
factory.setNamespaceAware(true);
factory.setValidating(true);

String location = "http://myserver/mycontent.xml";
try {
    builder = factory.newDocumentBuilder();
    Document document = builder.parse(location);
} catch (SAXException se) {
    // handle error
} catch (IOException ioe) {
    // handle error
} catch (ParserConfigurationException pce) {
    // handle error
}

An example of how one could pass the System property as a command line option is:
user.parserApp

4.6 XSLT Plugability

The XSLT Plugability classes allow an application programmer to obtain a Transformer object that is based on a specific XSLT stylesheet from a TransformerFactory implementation. In order to obtain a Transformer object, a programmer first obtains an instance of the TransformerFactory. The TransformerFactory instance is obtained via one of the static newInstance methods of the TransformerFactory class.

This method uses the following ordered lookup procedure to determine the TransformerFactory implementation class to load:

Use the javax.xml.transform.TransformerFactory system property

• Use the properties file "lib/jaxp.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the system property defined above. The jaxp.properties file is read only once by the JSR-000206 Java™API for XML Processing ("Specification") implementation and its values are then cached for future use. If the file does not exist when the first attempt is made to read from it, no further attempts are made to check for its existence. It is not possible to change the value of any property in jaxp.properties after it has been read for the first time.

• Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for the classname in the file META-INF/services/javax.xml.transform.TransformerFactory in jars available to the runtime.

• Platform default TransformerFactory instance.

If the TransformerFactory implementation class cannot be loaded or instantiated at runtime, a TransformerFactoryConfigurationException is thrown. This error message should contain a descriptive explanation of the problem and how the user can resolve it.

4.7 Examples

The following is a simple example of how to transform XML content:

Transformer transformer;
TransformerFactory factory = TransformerFactory.newInstance();
String stylesheet = "file:///home/user/mystylesheet.xsl";
String sourceId = "file:///home/user/sourcefile.xml";
try {
    transformer = factory.newTransformer(new StreamSource(stylesheet));
    transformer.transform(new StreamSource(sourceId), new StreamResult(System.out));
} catch (Exception e) {
    // handle error
}

The following example illustrates the serialization of a DOM node to an XML stream:

TransformerFactory tfactory = TransformerFactory.newInstance();
Transformer serializer = tfactory.newTransformer();
Properties oprops = new Properties();
oprops.put("method", "html");
oprops.put("indent-amount", "2");
serializer.setOutputProperties(oprops);
serializer.transform(new DOMSource(doc), new StreamResult(System.out));

Exceptions and Error Reporting

The following example illustrates the use of the URIResolver to resolve URIs to DOM nodes, in a transformation whose input is totally DOM based:

TransformerFactory tfactory = TransformerFactory.newInstance();
if (tfactory.getFeature(DOMSource.FEATURE) && tfactory.getFeature(StreamResult.FEATURE)) {
    DocumentBuilderFactory dfactory = DocumentBuilderFactory.newInstance();
    dfactory.setNamespaceAware(true); // Always, required for XSLT
    DocumentBuilder docBuilder = dfactory.newDocumentBuilder();
    // Set up to resolve URLs that correspond to our incl.xsl, // to a DOM node. Use an anonymous class for the URI resolver.
    final Node xslInc1 = docBuilder.parse("xsl/inc1/inc1.xsl");
    final Node xslInc2 = docBuilder.parse("xsl/inc2/inc2.xsl");
    tfactory.setURIResolver(new URIResolver() {
        public Source resolve(String href, String base)
            throws TransformerException {
            // ignore base.
            return (href.equals("inc1/inc1.xsl"))
                ? new DOMSource(xslInc1)
                : (href.equals("inc2/inc2.xsl"))
                    ? new DOMSource(xslInc2)
                    : null;
        }
    });
    // The TransformerFactory will call the anonymous URI
    // resolver set above when it encounters
    // &lt;xsl:include href="incl/inc1.xsl"&gt;
    Templates templates
        = tfactory.newTemplates(new DOMSource(docBuilder.parse(xslID), xslID));
    // Get a transformer from the templates.
    Transformer transformer = templates.newTransformer();
// Set up to resolve URLs that correspond to our foo2.xml, to
// a DOM node. Use an anonymous class for the URI resolver.
// Be sure to return the same DOM tree every time for the given URI.
final Node xmlSubdir1Foo2Node = docBuilder.parse("xml/subdir1/foo2.xml");
transformer.setURIResolver(new URIResolver() {
    public Source resolve(String href, String base)
        throws TransformerException {
        // ignore base because we're lazy, or we don't care.
        return (href.equals("subdir1/foo2.xml"))
            ? new DOMSource(xmlSubdir1Foo2Node)
            : null;
    }
});

// Now the transformer will call our anonymous URI resolver
// when it encounters the document("subdir1/foo2.xml") invocation.
transformer.transform(new DOMSource(docBuilder.parse(sourceID), sourceID),
                      new StreamResult(System.out));

The following example performs a transformation using DOM nodes as input for the TransformerFactory,
as input for the Transformer, and as the output of the transformation:

TransformerFactory tfactory = TransformerFactory.newInstance();
// Make sure the TransformerFactory supports the DOM feature.
if (tfactory.getFeature(DOMSource.FEATURE)
    && tfactory.getFeature(DOMResult.FEATURE)) {
    // Use javax.xml.parsers to create our DOMs.
    DocumentBuilderFactory dfactory = DocumentBuilderFactory.newInstance();
    dfactory.setNamespaceAware(true); // do this always for XSLT
    DocumentBuilder docBuilder = dfactory.newDocumentBuilder();

    // Create the Templates from a DOM.
    Node xslDOM = docBuilder.parse(xslID);
    DOMSource dsource = new DOMSource(xslDOM, xslID);
    Templates templates = tfactory.newTemplates(dsource);

    // Create the source tree in the form of a DOM.
    Node sourceNode = docBuilder.parse(sourceID);

    // Create a DOMResult that the transformation will fill in.
    DOMResult dresult = new DOMResult();

    // And transform from the source DOM tree to a result DOM tree.
    Transformer transformer = templates.newTransformer();
    transformer.transform(new DOMSource(sourceNode, sourceID), dresult);

    // The root of the result tree may now be obtained from
    // the DOMResult object.
    Node out = dresult.getNode();

    // Serialize it to System.out for diagnostics.
    Transformer serializer = tfactory.newTransformer();
    serializer.transform(new DOMSource(out), new StreamResult(System.out));
}
The following code fragment illustrates the use of the SAXSource and SAXResult objects:

```java
TransformerFactory tfactory = TransformerFactory.newInstance();
// Does this factory support SAX features?
if (tfactory.getFeature(SAXSource.FEATURE)
        && tfactory.getFeature(SAXResult.FEATURE)) {
    // Get a transformer.
    Transformer transformer = tfactory.newTransformer(new
        StreamSource(xslID));
    // Create a reader for reading.
    XMLReader reader = XMLReaderFactory.createXMLReader();
    transformer.transform(new SAXSource(reader, new InputSource(sourceID)),
        new SAXResult(new ExampleContentHandler()));
}
```

The following illustrates the feeding of SAX events from an `org.xml.sax.XMLReader` to a Transformer:

```java
TransformerFactory tfactory = TransformerFactory.newInstance();
// Does this factory support SAX features?
if (tfactory.getFeature(SAXTransformerFactory.FEATURE)) {
    // If so, we can safely cast.
    SAXTransformerFactory stfactory = ((SAXTransformerFactory) tfactory);
    // A TransformerHandler is a ContentHandler that will listen for
    // SAX events, and transform them to the result.
    TransformerHandler handler
        = stfactory.newTransformerHandler(new StreamSource(xslID));
    // Set the result handling to be a serialization to System.out.
    handler.setResult(new StreamResult(System.out));
    handler.getTransformer().setParameter("a-param", "hello to you!");
    // Create a reader, and set its content handler to be the
    TransformerHandler.
    XMLReader reader = XMLReaderFactory.createXMLReader();
    reader.setContentHandler(handler);
    // It's a good idea for the parser to send lexical events.
    // The TransformerHandler is also a LexicalHandler.
    reader.setProperty("http://xml.org/sax/properties/lexical-handler",
        handler);
    // Parse the source XML, and send the parse events to the
    TransformerHandler.
    reader.parse(sourceID);
}
```

The following code fragment illustrates the creation of a Templates object from SAX2 events sent from an XMLReader:

```java
TransformerFactory tfactory = TransformerFactory.newInstance();
// Does this factory support SAX features?
if (tfactory.getFeature(SAXTransformerFactory.FEATURE)) {
    // If so, we can safely cast.
    SAXTransformerFactory stfactory = ((SAXTransformerFactory) tfactory);
    // Have the factory create a special ContentHandler that will
    // create a Templates object.
```
TemplatesHandler handler = stfactory.newTemplatesHandler();

// If you don't do this, the TemplatesHandler won't know how to
// resolve relative URLs.
handler.setSystemId(xslID);

// Create a reader, and set its content handler to be the
// TemplatesHandler.
XMLReader reader = XMLReaderFactory.createXMLReader();
reader.setContentHandler(handler);

// Parse the source XML, and send the parse events to the
// TemplatesHandler.
reader.parse(xslID);

// Get the Templates reference from the handler.
Templates templates = handler.getTemplates();

// Ready to transform.
Transformer transformer = templates.newTransformer();
transformer.transform(new StreamSource(sourceID), new
StreamResult(System.out));

The following illustrates several transformations chained together. Each filter points to a parent
org.xml.sax.XMLReader , and the final transformation is caused by invoking org.xml.sax.XMLReader#parse on the final reader in the chain:

TransformerFactory tfactory = TransformerFactory.newInstance();
// Does this factory support SAX features?
if (tfactory.getFeature(SAXTransformerFactory.FEATURE)) {
    Templates stylesheet1 = tfactory.newTemplates(new StreamSource(xslID_1));
    Transformer transformer1 = stylesheet1.newTransformer();
    SAXTransformerFactory stf = (SAXTransformerFactory)tfactory;
    XMLReader reader = XMLReaderFactory.createXMLReader();
    XMLFilter filter1 = stf.newXMLFilter(new StreamSource(xslID_1));
    XMLFilter filter2 = stf.newXMLFilter(new StreamSource(xslID_2));
    XMLFilter filter3 = stf.newXMLFilter(new StreamSource(xslID_3));

    // transformer1 will use a SAX parser as its reader.
    filter1.setParent(reader);

    // transformer2 will use transformer1 as its reader.
    filter2.setParent(filter1);

    // transform3 will use transform2 as its reader.
    filter3.setParent(filter2);
    filter3.setContentHandler(new ExampleContentHandler());
    // filter3.setContentHandler(new org.xml.sax.helpers.DefaultHandler());
    // Now, when you call transformer3 to parse, it will set
    // itself as the ContentHandler for transform2, and
    // call transform2.parse, which will set itself as the
    // content handler for transform1, and call transform1.parse,
    // which will set itself as the content listener for the
    // SAX parser, and call parser.parse(new InputSource("xml/foo.xml"));

    // transformer1 will use a SAX parser as its reader.
    // transformer2 will use transformer1 as its reader.
    // transform3 will use transform2 as its reader.
    // filter3 will use filter1 as its reader.
    // filter3 will use filter2 as its reader.
    // filter3 will use filter3 as its reader.
    // Now, when you call transformer3 to parse, it will set
    // itself as the ContentHandler for transformer2, and
    // call transformer2.parse, which will set itself as the
    // content handler for transformer1, and call transformer1.parse,
    // which will set itself as the content listener for the
    // SAX parser, and call parser.parse(new InputSource("xml/foo.xml"));
The following code fragment illustrates the use of the stream Source and Result objects:

```java
// Create a TransformerFactory instance.
TransformerFactory tfactory = TransformerFactory.newInstance();
InputStream xslIS = new BufferedInputStream(new FileInputStream(xslID));
StreamSource xslSource = new StreamSource(xslIS);
// Note that if we don't do this, relative URLs cannot be resolved correctly!
xslSource.setSystemId(xslID);

// Create a transformer for the stylesheet.
Transformer transformer = tfactory.newTransformer(xslSource);
InputStream xmlIS = new BufferedInputStream(new FileInputStream(sourceID));
StreamSource xmlSource = new StreamSource(xmlIS);
// Note that if we don't do this, relative URLs cannot be resolved correctly!
xmlSource.setSystemId(sourceID);

// Transform the source XML to System.out.
transformer.transform( xmlSource, new StreamResult(System.out));
```

An example of how one could pass the System property as a command line option is:
```
java -Djavax.xml.transform.TransformerFactory=org.apache.xalan.processor.TransformerFactory-Impl user.parserApp
```

### 4.8 XPath Plugability

The XPath Plugability classes allow an application programmer to obtain an XPath object that is based on a specific object model implementation. In order to obtain an XPath object, a programmer first obtains an instance of the XPathFactory. The `XPathFactory` instance is obtained via one of the static `newInstance` methods of the `XPathFactory` class.

This method uses the following ordered lookup procedure to determine the `XPathFactory` implementation class to load:

- Use the `javax.xml.xpath.XPathFactory` system property
- Use the properties file "lib/jaxp.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the system property defined above. The jaxp.properties file is read only once by the JSR-000206 Java™API for XML Processing (“Specification”) implementation and its values are then cached for future use. If the file does not exist when the first attempt is made to read from it, no further attempts are made to check for its existence. It is not possible to change the value of any property in jaxp.properties after it has been read for the first time.
- Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for the classname in the file META-INF/services/javax.xml.xpath.XPath-Factory in jars available to the runtime.
- Platform default `XPathFactory` instance.

If the `XPathFactory` implementation class cannot be loaded or instantiated at runtime, a
4.9 Examples

The following example loads a document and evaluates the XPath expression “/widgets/widget[@name='a']/@quantity” against it.

```java
// parse the XML as a W3C Document
DocumentBuilder builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
org.w3c.Document document = builder.parse(new File("/widgets.xml"));

// evaluate the XPath expression against the Document
XPath xpath = XPathFactory.newInstance().newXPath();
String expression = "/widgets/widget[@name='a']/@quantity";
Double quantity = (Double) xpath.evaluate(expression, document, XPathConstants.NUMBER);
```

The evaluation of XPath expressions can return their results as one of five types: Node, NodeList, Boolean, Double, and String. The third argument to `evaluate` selects the result type. To get the `quantity` attribute as a string, use:

```java
String str_quantity = (String) xpath.evaluate(expression, document, XPathConstants.STRING);
```

This may seem somewhat clumsy, but it is necessary because the result type is not a property of the expression, it is a property of the context in which it is evaluated.

4.10 Validation Plugability

The Validation Plugability classes allow an application programmer to obtain a Schema Object that is based on a specific schema language implementation. In order to obtain a Schema Object, a programmer first obtains an instance of the SchemaFactory. The SchemaFactory instance is obtained via one of the static newInstance methods of the SchemaFactory Class.

This method uses the following ordered lookup procedure to determine the SchemaFactory implementation Class to load:

- If the system property `javax.xml.validation.SchemaFactory:schemaLanguage` is present (where `schemaLanguage` is the parameter to `newInstance`), then its value is read as a Class name. The method will try to create a new instance of this Class by using the ClassLoader, and returns it if it is successfully created.
- Read the properties file `$java.home/lib/jaxp.properties` in the JRE directory. This configuration file is in standard `java.util.Properties` format. `$java.home/lib/jaxp.properties` is read only once by the JSR-000206 Java™API for XML Processing (“Specification”) implementation and its values are then cached for future use. If the file does not exist when the first attempt is made to read from it, no further attempts are made to check for its existence. It is not possible to change the value of any property in `$java.home/lib/jaxp.properties` after it has been read for the first time.
If the property `javax.xml.validation.SchemaFactory:schemaLanguage` is present (where `schemaLanguage` is the parameter to `newInstance`), then its value is read as a `Class` name. The method will try to create a new instance of this `Class` by using the `ClassLoader`, and returns it if it is successfully created.

- The `ClassLoader` is asked for service provider provider-configuration files matching `javax.xml.validation.SchemaFactory` in the resource directory `META-INF/services`. See the JAR File Specification for file format and parsing rules. Each potential service provider is required to implement the method:

```java
public boolean isSchemaLanguageSupported(String schemaLanguage)
```

The first service provider found in `ClassLoader` order that supports the specified schema language is returned.


If all lookups fail, then an `IllegalArgumentException` will be thrown.

Tip

See `Properties.load(java.io.InputStream)` for exactly how a property file is parsed. In particular, colons `:` need to be escaped in a property file, so make sure schema language URIs are properly escaped. For example:

```
http://www.w3.org/2001/XMLSchema=org.acme.XSSchemaFactory
```

### 4.11 Streaming API for XML Plugability

JAXP defines a plugability mechanism to dynamically load compliant implementations of SAX and DOM parsers using the `javax.xml.parsers` and `javax.xml.transform` APIs. In an analogous manner, the StAX APIs define plugability mechanisms which allow applications to dynamically load compliant implementations of StAX.

See the discussion of `javax.xml.stream.XMLInputFactory`, `javax.xml.stream.XMLOutputFactory`, and `javax.xml.stream.XMLEventFactory` in the StAX Specification for further details.

### 4.12 Examples

The following example demonstrates how a streaming reader can be used to transform parts of a document.

Imagine, for the purposes of this somewhat contrived example, that you want to transform only those parts of your document that are labelled with the editorial status “draft”.

Begin by constructing a Transformer:

```java
XMLInputFactory inputFactory = XMLInputFactory.newInstance();
FileInputStream styleis = new FileInputStream("inStyle.xsl");
XMLStreamReader stylesheet = inputFactory.createXMLStreamReader(styleis);
XMLStreamReader stylesheet = inputFactory.createXMLStreamReader(styleis);
```
Source style = new StAXSource(styleSheet);
TransformerFactory tf = TransformerFactory.newInstance();
Transformer t = tf.newTransformer(style);

Then create a reader for your input file:

FileInputStream fis = new FileInputStream("inFile.xml");
XMLStreamReader reader = inputFactory.createXMLStreamReader(fis);

Finally, walk through the input document using the pull API. Each time you find an element marked “draft”, transform it:

while(reader.hasNext()) {
    if (reader.getEventType() == XMLStreamReader.START_ELEMENT) {
        if ("draft".equals(reader.getAttributeValue(null, "status"))) {
            // Transform this...
            Source source = new StAXSource(reader);
            DOMResult result = new DOMResult();
            t.transform(source, result);
            // Do something with the result...
        } else {
            // This element doesn't need to be transformed
        }
    }
    reader.next();
}

Of course, it's also possible to simply transform entire documents using the event interfaces.

4.13 Datatype Plugability

The Datatype Plugability classes allow an application programmer to obtain a variety of datatype objects. In order to obtain a datatype, a programmer first obtains an instance of the DatatypeFactory. The DatatypeFactory instance is obtained via one of the static newInstance methods of the DatatypeFactory class.

This method uses the following ordered lookup procedure to determine the DatatypeFactory implementation Class to load:

1. If the system property specified by DATATYPEFACTORY_PROPERTY, "javax.xml.datatype.DatatypeFactory", exists, a class with the name of the property's value is instantiated. Any Exception thrown during the instantiation process is wrapped as a DatatypeConfigurationException.

2. If the file ${JAVA_HOME}/lib/jaxp.properties exists, it is loaded in a Properties Object. The Properties Object is then queried for the property as documented in the prior step and processed as documented in the prior step.

3. The services resolution mechanism is used, e.g. META-INF/services/java.xml.datatype.DatatypeFactory. Any Exception thrown during the instantiation process is wrapped as a DatatypeConfigurationException.
4. The final mechanism is to attempt to instantiate the Class specified by 
DATATYPEFACTORY_IMPLEMENTATION_CLASS. Any Exception thrown during the instantiation 
process is wrapped as a DatatypeConfigurationException.

The DATATYPEFACTORY_IMPLEMENTATION_CLASS constant is intended for JAXP implementors 
to specify a “class of last resort”. Application programmers should never refer to the constant in their 
programs. Instead, they should rely on the lookup procedure to locate an appropriate class.

4.14 Thread Safety

Implementations of SAXParser, DocumentBuilder, Transformer, Validator, ValidatorHandler, 
XPath, and XPathExpression are not expected to be thread safe by this specification. This means that application programmers should not expect to be able to use an instance of any of them 
in more than one thread at a time without side effects. If a programmer is creating a multi-threaded application, 
they should make sure that only one thread has access to any instance at any given time.

Configuration of a SAXParserFactory, DocumentBuilderFactory, TransformerFactory, 
SchemaFactory, or XPathFactory is also not expected to be thread safe. This means that an application 
programmer should not allow an instance of any of these factories to have its setter methods accessed 
from more than one thread.

It is expected that the newSAXParser method of a SAXParserFactory implementation, the new- 
DocumentBuilder method of a DocumentBuilderFactory and the newTransformer method 
of a TransformerFactory will be thread safe without side effects. This means that an application 
programmer should expect to be able to create transformer instances in multiple threads at once from a 
shared factory without side effects or problems. Note however that this guarantee can not automatically 
be extended to all the classes that a transformation might access. Consider, for example, the URIResolver 
or ErrorHandler classes. If an application is going to share instances of these classes across several 
transformation threads, the application must assure that they are thread-safe.

The Schema class is thread safe.

4.15 Properties For Enabling Schema Validation

class javax.xml.parsers.SAXParserFactory

The validating property must have the value true for any of the property strings defined below to take effect. 
Otherwise, the values of the properties defined below will be ignored. This value can be set by invoking

setValidating(true)
class javax.xml.parsers.SAXParser

The setProperty method in SAXParser must support the property strings defined below to indicate the 
schema language and the source of the schema file(s) to the parser:

http://java.sun.com/xml/jaxp/properties/schemaLanguage

This property defines the schema language to be used for validation. The value of this property must be 
the URI of the schema language specification. To be compliant with this version of the specification, the 

When setValidating is set to true and a schema language is set, then the parser must validate against that
schema language only. For example if an application sets the schemaLanguage property to XML Schemas then the parser must try to validate against the XML schema only, even if the document has a DOCTYPE declaration that refers to a DTD.

http://java.sun.com/xml/jaxp/properties/schemaSource

The XML Schema Recommendation explicitly states that the inclusion of schemaLocation / noNamespaceSchemaLocation attributes in an instance document is only a hint; it does not mandate that these attributes must be used to locate schemas.

The schemaSource property lets the user set the schema(s) to validate against. If the target namespace of a schema specified using this property matches the target namespace of a schema occurring in schemaLocation attribute, the schema specified by the user using this property will be used and the instance document's schemaLocation attribute will be effectively ignored. However if the target namespace of any schema specified using this property doesn't match the target namespace of a schema occurring in the instance document, then the hint specified in the instance document will be used for validation. The acceptable value for this property must be one of the following:

• URI of the schema as a String
• InputStream with the contents of the schema
• SAX InputSource
• File
• an array of Objects with the contents being one of the types defined above. An array of Objects can be used only when the schema language has the ability to assemble a schema at runtime. When an array of Objects is passed it is illegal to have two schemas that share the same namespace.

If no target namespace is defined, then only one schema can be referenced by the property and it must work exactly the way xsi:noNamespaceSchemaLocation does.

It is illegal to set the schemaSource property if the schemaLanguage property has not been set. In that case, the implementation must throw a SAXNotSupportedException with a detailed message.

If the schemaSource property is set using a String, the parser must pass the value of the property to the org.xml.sax.EntityResolver with the publicId set to null.

javax.xml.parsers.DocumentBuilderFactory

The same property strings as described above for the SAXParser must be supported by DocumentBuilderFactory.

setAttribute method.

When setValidating is set to true and a schema language is set then the parser must validate against that schema language only. For example if an application sets the schema language property to XML Schemas the parser must try to validate against the XML schema only, even if the document has a DOCTYPE declaration that refers to a DTD.

It is illegal to set the schemaSource property if the schemaLanguage property has not been set. In that case, the implementation must throw an IllegalArgumentException with a detailed message.

Note: None of the properties will take effect till the setValidating(true) has been called on the SAXParser-
Factory or the DocumentBuilderFactory that was used to create the SAXParser or the DocumentBuilder.

The table below shows the results of various configuration scenarios. In all cases, we assume that setValidating(true) has been called.

The table below shows the results of various configuration scenarios. In all cases, we assume that setValidating(true) has been called.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>error as per JAXP 1.1 spec. Must have a DOCTYPE declaration when validation is turned on.</td>
<td>N/A</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>Error. Schema language must be set.</td>
<td>N/A</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes / no</td>
<td>Error. Schema language must be set.</td>
<td>N/A</td>
</tr>
<tr>
<td>yes / no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>xml schemas</td>
<td>schema files referred to using the schema location attributes present in the instance document</td>
</tr>
<tr>
<td>yes / no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>xml schemas</td>
<td>schema file referred to in the schemaSource property</td>
</tr>
<tr>
<td>yes / no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>xml schemas</td>
<td>schema file referred to in the schema source property, if the target namespace matches. The schema file referred to in the schema location attribute is ignored only if the target namespace matches</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes / no</td>
<td>DTD</td>
<td>DTD referred to in the DOCTYPE</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes / no</td>
<td>Error. Schema source cannot be set without setting the schema language.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

4.16 Samples Using the Properties

Sax parser sample:

```java
try {
    SAXParserFactory spf = SAXParserFactory.newInstance();
```
spf.setNamespaceAware(true);
spf.setValidating(true);
SAXParser sp = spf.newSAXParser();
sp.setProperty("http://java.sun.com/xml/jaxp/properties/schemaLanguage",
"http://www.w3.org/2001/XMLSchema");
sp.setProperty("http://java.sun.com/xml/jaxp/properties/schemaSource",
"http://www.example.com/Report.xsd");
DefaultHandler dh = new DefaultHandler();
sp.parse("http://www.example.com/foo.xml", dh);
} catch(Exception e) {
    e.printStackTrace();
}

DOM parser sample:

try {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
dbf.setValidating(true);
dbf.setAttribute(
        "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
        "http://www.w3.org/2001/XMLSchema");
        "http://www.example.com/Report.xsd");
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse("http://www.example.com/foo.xml");
} catch(Exception e) {
    e.printStackTrace();
}

4.17 Recommended Implementation of Properties

It is recommended that parser implementations recognize properties defined in the form of a URI, as above. Such implementations avoid conflicts in the use of the feature and property strings among parser implementations. That is also the way in which SAX defines feature and property strings.

5. Conformance Requirements

This section describes the conformance requirements for implementations of this specification. Implementations that are accessed via the APIs defined here must implement these constraints, without exception, to provide a predictable environment for application development and deployment.

Note that applications may provide non-conformant implementations that are able to support the plugability mechanism defined in the specification, however the system default processor must meet the conformance requirements defined below.

All Implementations of JSR-000206 Java™API for XML Processing ("Specification") Must Support the Following Specifications:

• Namespaces in XML 1.1 [http://www.w3.org/TR/xml-names11/], Namespaces in XML 1.0 [spec.namespaces-1.0.link;], Namespaces in XML 1.0 Errata [spec.namespaces-1.0-errata.link;]


• XSL Transformations (XSLT) Version 1.0 [http://www.w3.org/TR/xslt], XSL Transformations (XSLT) Version 1.0 Errata [http://www.w3.org/1999/11/REC-xslt-19991116-errata]

• XML Path Language (XPath) Version 1.0 [http://www.w3.org/TR/xpath], XML Path Language (XPath) Version 1.0 Errata [http://www.w3.org/1999/11/REC-xpath-19991116-errata]

• XML Inclusions (XInclude) Version 1.0 [http://www.w3.org/TR/xinclude/]

• Document Object Model (DOM) Level 2 Core [http://www.w3.org/TR/DOM-Level-2-Core], Document Object Model (DOM) Level 2 Traversal and Range, Document Object Model (DOM) Level 2 Events

• Document Object Model (DOM) Level 3 Core [http://www.w3.org/TR/DOM-Level-3-Core], Document Object Model (DOM) Level 3 Load and Save

• Simple API for XML (SAX) 2.0.2 (sax2r3) [http://sax.sourceforge.net/], http://sax.sourceforge.net/?selected=ext
6. XML Inclusions (XInclude)

6.1 What is XML Inclusions (XInclude)

JSR-000206 Java™ API for XML Processing (“Specification”) supports XInclude as defined in XML Inclusions (XInclude) Version 1.0 [http://www.w3.org/TR/xinclude/].

XInclude specifies a processing model and syntax for general purpose inclusion. Inclusion is accomplished by merging a number of XML information sets into a single composite Infoset. Specification of the XML documents (infosets) to be merged and control over the merging process is expressed in XML-friendly syntax (elements, attributes, URI references).

— XML Inclusions (XInclude) Version 1.0, Abstract [http://www.w3.org/TR/xinclude/#abstract]

6.2 Implementation Required by JSR-000206 Java™ API for XML Processing (“Specification”)

JSR-000206 Java™ API for XML Processing (“Specification”) implements XML Inclusions (XInclude) Version 1.0 with the following limitations

• at the time this specification was developed, there was no standard fragment identifier syntax for “application/xml” resources
• supports XPointers using the XPointer Framework and XPointer element() scheme
• no other XPointer schemes or addressing mechanisms are supported and it is an error to use an other XPointer scheme or addressing mechanism, the error is signaled by throwing a java.lang.Exception for DOM processing and an org.xml.sax.SAXParseException for SAX processing

XInclude processing defaults to false. See Javadoc for javax.xml.parsers.DocumentBuilderFactory and javax.xml.parsers.SAXParserFactory for methods to enable/disable/query the state of XInclude processing.

7.1 JSR-000206 Java™ API for XML Processing (“Specification”) Specification Version Information

JSR-000206 Java™ API for XML Processing (“Specification”) specification version information is made available via java.lang.Package methods

- public String getSpecificationTitle();
  
  Return the title of the specification that this package implements. null is returned if it is not known.

- public String getSpecificationVersion();
  
  Returns the version number of the specification that this package implements. This version string must be a sequence of positive decimal integers separated by "."s and may have leading zeros. When version strings are compared the most significant numbers are compared. null is returned if it is not known.

- public String getSpecificationVendor();
  
  Returns the name of the organization, vendor, or company that owns and maintains the specification of the classes that implement this package. null is returned if it is not known.

This version information is retrieved and made available by the ClassLoader instance that loaded the class(es). Typically, it is stored in the manifest that is distributed with the classes. Implementations are required to provide this information with the following values:

<table>
<thead>
<tr>
<th>Specification Title</th>
<th>JSR-000206 Java™ API for XML Processing (“Specification”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification Vendor</td>
<td>Sun Microsystems, Inc.</td>
</tr>
<tr>
<td>Specification Version</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Sample META-INF/MANIFEST.MF entries to provide this information would be:

```
Specification-Title : JSR-000206 Java™ API for XML Processing (“Specification”)
Specification-Vendor : Sun Microsystems, Inc.
Specification-Version : 1.4
```

JSR-000206 Java™ API for XML Processing (“Specification”) Specification
7.2 JSR-000206 Java™ API for XML Processing (“Specification”) Implementation Version Information

JSR-000206 Java™ API for XML Processing (“Specification”) implementation version information is made available via java.lang.Package methods

- public String getImplementationTitle();
  Return the title of this package. null is returned if it is not known.

- public String getImplementationVersion();
  Returns the version of this implementation. It consists of any string assigned by the vendor of this implementation and does not have any particular syntax specified or expected by the Java runtime. It may be compared for equality with other package version strings used for this implementation by this vendor for this package. null is returned if it is not known.

- public String getImplementationVendor();
  Returns the name of the organization, vendor or company that provided this implementation.

This version information is retrieved and made available by the ClassLoader instance that loaded the class(es). Typically, it is stored in the manifest that is distributed with the classes. Implementations are required to provide this information with reasonable values:

Implementation Title
Implementation Vendor
Implementation Version

Sample META-INF/MANIFEST.MF entries to provide this information would be:

Implementation-Title : My Implementation Title
Implementation-Vendor : My Implementation Vendor
Implementation-Version : My Implementation Version

See the JAR File Specification, ${JAVA_HOME}/docs/guide/jar/jar.html, for detailed format information.
8. Constants

Table 8.1. Constants in javax.xml.datatype.DatatypeFactory

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>DATATYPEFACTORY_PROPERTY</td>
<td>&quot;javax.xml.datatype.DatatypeFactory&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>DATATYPEFACTORY_IMPLEMENTATION_CLASS</td>
<td>Implementation defined^</td>
</tr>
</tbody>
</table>

^Implementers should specify the name of an appropriate class to be instantiated if no other implementation resolution mechanism succeeds. Users should not refer to this field; it is intended only to document a factory implementation detail.

Table 8.2. Constants in javax.xml.datatype.DatatypeConstants

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>JANUARY</td>
<td>1</td>
</tr>
<tr>
<td>int</td>
<td>FEBRUARY</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>MARCH</td>
<td>3</td>
</tr>
<tr>
<td>int</td>
<td>APRIL</td>
<td>4</td>
</tr>
<tr>
<td>int</td>
<td>MAY</td>
<td>5</td>
</tr>
<tr>
<td>int</td>
<td>JUNE</td>
<td>6</td>
</tr>
<tr>
<td>int</td>
<td>JULY</td>
<td>7</td>
</tr>
<tr>
<td>int</td>
<td>AUGUST</td>
<td>8</td>
</tr>
<tr>
<td>int</td>
<td>SEPTEMBER</td>
<td>9</td>
</tr>
<tr>
<td>int</td>
<td>OCTOBER</td>
<td>10</td>
</tr>
<tr>
<td>int</td>
<td>NOVEMBER</td>
<td>11</td>
</tr>
<tr>
<td>int</td>
<td>DECEMBER</td>
<td>12</td>
</tr>
<tr>
<td>int</td>
<td>LESSER</td>
<td>-1</td>
</tr>
<tr>
<td>int</td>
<td>EQUAL</td>
<td>0</td>
</tr>
<tr>
<td>int</td>
<td>GREATER</td>
<td>1</td>
</tr>
<tr>
<td>int</td>
<td>INDETERMINATE</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>FIELD_UNDEFINED</td>
<td>Integer.MIN_VALUE</td>
</tr>
<tr>
<td>javax.xml.datatype.DatatypeConstants.FIELD</td>
<td>YEARS</td>
<td>new Field(&quot;YEARS&quot;, 0)</td>
</tr>
<tr>
<td>javax.xml.datatype.DatatypeConstants.FIELD</td>
<td>MONTHS</td>
<td>new Field(&quot;MNTHS&quot;, 1)</td>
</tr>
<tr>
<td>javax.xml.datatype.DatatypeConstants.FIELD</td>
<td>DAYS</td>
<td>new Field(&quot;DAYS&quot;, 2)</td>
</tr>
<tr>
<td>javax.xml.datatype.DatatypeConstants.FIELD</td>
<td>HOURS</td>
<td>new Field(&quot;HOURS&quot;, 3)</td>
</tr>
<tr>
<td>javax.xml.datatype.DatatypeConstants.FIELD</td>
<td>MINUTES</td>
<td>new Field(&quot;MINUTES&quot;, 4)</td>
</tr>
<tr>
<td>javax.xml.datatype.DatatypeConstants.FIELD</td>
<td>SECONDS</td>
<td>new Field(&quot;SECONDS&quot;, 5)</td>
</tr>
<tr>
<td>Type</td>
<td>Name</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>DATETIME</td>
<td>new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, &quot;dateTime&quot;)</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>TIME</td>
<td>new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, &quot;time&quot;)</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>DATE</td>
<td>new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, &quot;date&quot;)</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>GYEARMONTH</td>
<td>new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, &quot;gYearMonth&quot;)</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>GMONTHDAY</td>
<td>new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, &quot;gMonthDay&quot;)</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>GYEAR</td>
<td>new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, &quot;gYear&quot;)</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>GMONTH</td>
<td>new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, &quot;gMonth&quot;)</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>GDAY</td>
<td>new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, &quot;gDay&quot;)</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>DURATION</td>
<td>new QName(XMLConstants.W3C_XML_SCHEMA_NS_URI, &quot;duration&quot;)</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>DARATION_DAYTIME</td>
<td>new QName(XMLConstants.W3C_XPATH_DATATYPE_NS_URI, &quot;dayTimeDuration&quot;)</td>
</tr>
<tr>
<td>javax.xml.namespace.QName</td>
<td>DARATION_YEARMONTH</td>
<td>new QName(XMLConstants.W3C_XPATH_DATATYPE_NS_URI, &quot;yearMonthDuration&quot;)</td>
</tr>
<tr>
<td>int</td>
<td>MAX_TIMEZON_OFFSET</td>
<td>-14 * 60</td>
</tr>
<tr>
<td>int</td>
<td>MIN_TIMEZON_OFFSET</td>
<td>14 * 60</td>
</tr>
</tbody>
</table>

Table 8.3. Constants in javax.xml.transform.dom.DOMResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>FEATURE</td>
<td>&quot;<a href="http://javax.xml.transform.dom.DOMResult/feature">http://javax.xml.transform.dom.DOMResult/feature</a>&quot;</td>
</tr>
<tr>
<td>Type</td>
<td>Name</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>FEATURE</td>
<td>&quot;<a href="http://javax.xml.transform.dom.DOMSource/feature">http://javax.xml.transform.dom.DOMSource/feature</a>&quot;</td>
</tr>
</tbody>
</table>

**Table 8.5. Constants in javax.xml.transform.sax.SAXResult**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>FEATURE</td>
<td>&quot;<a href="http://javax.xml.transform.sax.SAXResult/feature">http://javax.xml.transform.sax.SAXResult/feature</a>&quot;</td>
</tr>
</tbody>
</table>

**Table 8.6. Constants in javax.xml.transform.sax.SAXSource**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>FEATURE</td>
<td>&quot;<a href="http://javax.xml.transform.sax.SAXSource/feature">http://javax.xml.transform.sax.SAXSource/feature</a>&quot;</td>
</tr>
</tbody>
</table>

**Table 8.7. Constants in javax.xml.transform.sax.SAXTransformerFactory**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>FEATURE_XMLFILTER</td>
<td>&quot;<a href="http://javax.xml.transform.sax.SAXTransformerFactory/feature/xmlfilter">http://javax.xml.transform.sax.SAXTransformerFactory/feature/xmlfilter</a>&quot;</td>
</tr>
</tbody>
</table>

**Table 8.8. Constants in javax.xml.transform.stream.StreamResult**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
</table>

**Table 8.9. Constants in javax.xml.transform.stream.StreamSource**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
</table>
Table 8.10. Constants in `javax.xml.transform.OutputKeys`

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>METHOD</td>
<td>&quot;method&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>VERSION</td>
<td>&quot;version&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>ENCODING</td>
<td>&quot;encoding&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>OMIT_XML_DECLARATION</td>
<td>&quot;omit-xml-declaration&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>STANDALONE</td>
<td>&quot;standalone&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>DOCTYPE_PUBLIC</td>
<td>&quot;doctype-public&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>DOCTYPE_SYSTEM</td>
<td>&quot;doctype-system&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>CDATA_SECTION_ELEMENTS</td>
<td>&quot;cdata-section-elements&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>INDENT</td>
<td>&quot;indent&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>MEDIA_TYPE</td>
<td>&quot;media-type&quot;</td>
</tr>
</tbody>
</table>

Table 8.11. Constants in `javax.xml.transform.Result`

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>PI_DISABLE_OUTPUT_ESCAPING</td>
<td>&quot;javax.xml.transform.disable-output-escaping&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>PI_ENABLE_OUTPUT_ESCAPING</td>
<td>&quot;javax.xml.transform.enable-output-escaping&quot;</td>
</tr>
</tbody>
</table>

Table 8.12. Constants in `javax.xml.transform.stax.StAXResult`

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
</table>

Table 8.13. Constants in `javax.xml.transform.stax.StAXSource`

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
</table>
Table 8.14. Constants in `javax.xml.xpath.XPathConstants`

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>javax.xml.namespace.QName</code></td>
<td>NUMBER</td>
<td>New QName(&quot;<a href="http://www.w3.org/1999/XSL/Transform">http://www.w3.org/1999/XSL/Transform</a>&quot;, &quot;NUMBER&quot;)</td>
</tr>
<tr>
<td><code>javax.xml.namespace.QName</code></td>
<td>STRING</td>
<td>new QName(&quot;<a href="http://www.w3.org/1999/XSL/Transform">http://www.w3.org/1999/XSL/Transform</a>&quot;, &quot;STRING&quot;)</td>
</tr>
<tr>
<td><code>javax.xml.namespace.QName</code></td>
<td>BOOLEAN</td>
<td>new QName(&quot;<a href="http://www.w3.org/1999/XSL/Transform">http://www.w3.org/1999/XSL/Transform</a>&quot;, &quot;BOOLEAN&quot;)</td>
</tr>
<tr>
<td><code>javax.xml.namespace.QName</code></td>
<td>NODESET</td>
<td>new QName(&quot;<a href="http://www.w3.org/1999/XSL/Transform">http://www.w3.org/1999/XSL/Transform</a>&quot;, &quot;NODESET&quot;)</td>
</tr>
<tr>
<td><code>javax.xml.namespace.QName</code></td>
<td>NODE</td>
<td>new QName(&quot;<a href="http://www.w3.org/1999/XSL/Transform">http://www.w3.org/1999/XSL/Transform</a>&quot;, &quot;NODE&quot;)</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td>DOM_OBJECT_MODEL</td>
<td>&quot;<a href="http://java.sun.com/jaxp/xpath/dom">http://java.sun.com/jaxp/xpath/dom</a>&quot;</td>
</tr>
</tbody>
</table>

Table 8.15. Constants in `javax.xml.xpath.XPathFactory`

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.String</code></td>
<td>DEFAULT_PROPERTY_NAME</td>
<td>&quot;javax.xml.xpath.XPathFactory&quot;</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td>DEFAULT_OBJECT_MODEL_URI</td>
<td>&quot;<a href="http://java.sun.com/jaxp/xpath/dom">http://java.sun.com/jaxp/xpath/dom</a>&quot;</td>
</tr>
</tbody>
</table>

Table 8.16. Constants in `javax.xml.stream.XMLInputFactory`

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.String</code></td>
<td>IS_NAMESPACE_AWARE</td>
<td>&quot;javax.xml.stream.isNamespaceAware&quot;</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td>IS_VALIDATING</td>
<td>&quot;javax.xml.stream.isValidating&quot;</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td>IS_COALESCEING</td>
<td>&quot;javax.xml.stream.isCoalescing&quot;</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td>IS_REPLACING_ENTITY_REFERENCES</td>
<td>&quot;javax.xml.stream.isReplacingEntityReferences&quot;</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td>IS_SUPPORTING_EXTERNAL_ENTITIES</td>
<td>&quot;javax.xml.stream.isSupportingExternalEntities&quot;</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td>SUPPORT_DTD</td>
<td>&quot;javax.xml.stream.supportDTD&quot;</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td>REPORTER</td>
<td>&quot;javax.xml.stream.reporter&quot;</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td>RESOLVER</td>
<td>&quot;javax.xml.streamresolver&quot;</td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td>ALLOCATOR</td>
<td>&quot;javax.xml.stream.alloc&quot;</td>
</tr>
</tbody>
</table>

Table 8.17. Constants in `javax.xml.stream.XMLOutputFactory`

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.lang.String</code></td>
<td>IS_REPAIRING_NAMESPACES</td>
<td>&quot;javax.xml.stream.isRepairingNamespaces&quot;</td>
</tr>
</tbody>
</table>
**Table 8.18. Constants in javax.xml.stream.XMLStreamConstants**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>START_ELEMENT</td>
<td>1</td>
</tr>
<tr>
<td>int</td>
<td>END_ELEMENT</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>PROCESSING_INSTRUCTION</td>
<td>3</td>
</tr>
<tr>
<td>int</td>
<td>CHARACTERS</td>
<td>4</td>
</tr>
<tr>
<td>int</td>
<td>COMMENT</td>
<td>5</td>
</tr>
<tr>
<td>int</td>
<td>SPACE</td>
<td>6</td>
</tr>
<tr>
<td>int</td>
<td>START_DOCUMENT</td>
<td>7</td>
</tr>
<tr>
<td>int</td>
<td>END_DOCUMENT</td>
<td>8</td>
</tr>
<tr>
<td>int</td>
<td>ENTITY_REFERENCE</td>
<td>9</td>
</tr>
<tr>
<td>int</td>
<td>ATTRIBUTE</td>
<td>10</td>
</tr>
<tr>
<td>int</td>
<td>DTD</td>
<td>11</td>
</tr>
<tr>
<td>int</td>
<td>CDATA</td>
<td>12</td>
</tr>
<tr>
<td>int</td>
<td>NAMESPACE</td>
<td>13</td>
</tr>
<tr>
<td>int</td>
<td>NOTATION_DECLARATION</td>
<td>14</td>
</tr>
<tr>
<td>int</td>
<td>ENTITY_DECLARATION</td>
<td>15</td>
</tr>
</tbody>
</table>

**Table 8.19. Constants in javax.xml.XMLConstants**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>NULL_NS_URI</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>DEFAULT_NS_PREFIX</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>XML_NS_PREFIX</td>
<td>&quot;xml&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>XMLNS_ATTRIBUTE_NS_URI</td>
<td>&quot;<a href="http://www.w3.org/2000/xmlns/">http://www.w3.org/2000/xmlns/</a>&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>XMLNS_ATTRIBUTE</td>
<td>&quot;xmlns&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>W3C_XML_SCHEMA_NS_URI</td>
<td>&quot;<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>W3C_XML_SCHEMA_INSTANCE_NS_URI</td>
<td>&quot;<a href="http://www.w3.org/2001/XMLSchemainstance">http://www.w3.org/2001/XMLSchemainstance</a>&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>W3C_XPATH_DATATYPE_NS_URI</td>
<td>&quot;<a href="http://www.w3.org/2003/11/xpath-datatypes">http://www.w3.org/2003/11/xpath-datatypes</a>&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>XML_DTD_NS_URI</td>
<td>&quot;<a href="http://www.w3.org/TR/REC-xml">http://www.w3.org/TR/REC-xml</a>&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>RELAXNG_NS_URI</td>
<td>&quot;<a href="http://relaxng.org/ns/structure/1.0">http://relaxng.org/ns/structure/1.0</a>&quot;</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>FEATURE_SECURE_PROCESSING</td>
<td>&quot;<a href="http://javax.xml.XMLConstants/feature/secure-processing">http://javax.xml.XMLConstants/feature/secure-processing</a>&quot;</td>
</tr>
</tbody>
</table>
9. Changes Since JAXP 1.3

The following sections summarize the significant changes introduced in JAXP 1.4. For additional details, please see the appropriate JavaDoc.

9.1 Package javax.xml.datatype

Class DatatypeFactory


- Added newInstance(String, ClassLoader) method.

Class Duration

- Return false if the argument to equals() is null instead of throwing NullPointerException.

Class XMLGregorianCalendar

- Return false if the argument to equals() is null instead of throwing NullPointerException.

9.2 Package javax.xml.parsers

Class DocumentBuilderFactory

- Added newInstance(String, ClassLoader) method.

Class SAXParserFactory

- Added newInstance(String, ClassLoader) method.

9.3 Package javax.xml.stream

Added the javax.xml.stream package to support StAX. The package consists of the following interfaces:

- EventFilter
- Location
- StreamFilter
- XMLEventReader
- XMLEventWriter
- XMLReporter
- XMLResolver
- XMLStreamConstants
- XMLStreamReader
- XMLStreamWriter

Classes:

- XMLEventFactory
- XMLInputFactory
- XMLOutputFactory

Exceptions:
And errors:
  • FactoryConfigurationException

9.4 Package javax.xml.stream.events

Added the javax.xml.stream.events package to support StAX. The package consists of the following interfaces:
  • Attribute
  • Characters
  • Comment
  • DTD
  • EndDocument
  • EndElement
  • EntityDeclaration
  • EntityReference
  • Namespace
  • NotationDeclaration
  • ProcessingInstruction
  • StartDocument
  • StartElement
  • XMLEvent

9.5 Package javax.xml.stream.util

Added the javax.xml.stream.util package to support StAX. The package consists of the following interfaces:
  • XMLEventAllocator
  • XMLEventConsumer

And classes:
  • EventReaderDelegate
  • StreamReaderDelegate

9.6 Package javax.xml.transform

Class ErrorListener
  • Clarified the semantics of fatalError().

Class Transformer
  • Clarified the semantics of getOutputProperty() with respect to properties that have not been set explicitly with either setOutputProperty() or xsl:output in the stylesheet.

Class TransformerFactory
  • Added newInstance(String, ClassLoader) method.
9.7 Package javax.xml.transform.stax

Added the javax.xml.transform.stax package to support StAX. The package consists of the following classes:

- StAXResult
- StAXSource

9.8 Package javax.xml.validation

**Class Schema**

- Clarified that the features set on the SchemaFactory should be passed to the Validator created with newValidator().
- Clarified that the features set on the SchemaFactory should be passed to the ValidatorHandler created with newValidatorHandler().

**Class SchemaFactory**

- Added newInstance(String, ClassLoader) method.
- Clarified that the features set on the SchemaFactory should be passed to the Schemas created with newSchema(). Included a note to implementors and developers about the subtleties associated with newSchema() in this context.
- Clarified that the inputs to newSchema(Source[]) are expected to be XML documents or elements.
- Updated documentation of setFeature() to clarify that the features set on the SchemaFactory should be passed to the Schemas created with this factory and by extension to the Validator and ValidatorHandlers created from that Schema.

**Class SchemaFactoryLoader**

The SchemaFactoryLoader class was created during the JAXP 1.3 development process. Shortly before JAXP 1.3 was finished, the factory mechanisms associated with validation were changed, and this class was removed. Unfortunately, the file was left in a repository and it slipped into the Java 2 Platform Standard Edition 5.0 API Specification.

Since the class can neither be added to JAXP 1.3 nor removed from the Java 5.0 API Specification due to backwards compatibility issues, it is being added to JAXP 1.4. The class is harmless and should not be used. It is being added simply to avoid the confusion that arises when developers notice that it's defined in the platform but not in JAXP.

*Do not use this class.*

**Class TypeInfoProvider**

Extended the semantics of getElementTypeInfo(), allowing it to be called from either the startElement event or the endElement event. This allows the API to support W3C XML Schema union types more usefully.

When W3C XML Schema validation is being performed, in the case where an element has a union type, the TypeInfo returned by a call to getElementTypeInfo() from the startElement event will be the union type. The TypeInfo returned by a call from the endElement event will be the actual member type used to validate the element.
Class Validator
- The Validator may now accept Sources other than DOMSource and SAXSource; for example, StAXSource.
- Clarified that the inputs to validate() are expected to be XML documents or elements.

9.9 Package javax.xml.xpath

Class XPath
- Clarified that it is the XPathVariableResolver in effect at compile time that is used to resolve any variables that appear in the expression.

Class XPathFactory
- Added newInstance(String, ClassLoader) method.
10. Changes Since JAXP 1.4

10.1 New properties

Three new JAXP properties along with their corresponding System Properties are added to specify the type of external connections that can or cannot be permitted. The property values are a list of protocols. The JAXP processors should check if a given external connection is allowed by matching the protocol with those in the list. Processors may attempt to establish the connection if it is on the list, or reject it if not.

10.2 Definition

1. New JAXP Properties

1) http://javax.xml.XMLConstants/property/accessExternalDTD: restrict access to external DTDs, external Entity References to the protocols specified. The parser should check the protocol of a connection URL against the value of this property before it attempts to make connection to resolve any external DTDs. If the protocol that the connection is attempted to is listed in the value of the property, the connection is allowed. Otherwise, it should be rejected with a runtime exception.

2) http://javax.xml.XMLConstants/property/accessExternalSchema: restrict access to the protocols specified for external reference set by the schemaLocation attribute, Import and Include element. The schema parser should check the protocol of a connection URL against the value of this property before it attempts to make connection to resolve any external schemas. If the protocol that the connection is attempted to is listed in the value of the property, the connection is allowed. Otherwise, it should be rejected with a runtime exception.

3) http://javax.xml.XMLConstants/property/accessExternalStylesheet: restrict access to the protocols specified for external reference set by the stylesheet processing instruction, document function, Import and Include element. The parser of the XSL transformer should check the protocol of a connection URL against the value of this property before it attempts to make connection to resolve any external stylesheets. If the protocol that the connection is attempted to is listed in the value of the property, the connection is allowed. Otherwise, it should be rejected with a runtime exception.

2. System Properties corresponding to the JAXP properties above

1) javax.xml.accessExternalDTD: same as accessExternalDTD.

2) javax.xml.accessExternalSchema: same as accessExternalSchema.
3) javax.xml.accessExternalStylesheet: same as accessExternalStylesheet.

3. <Java Home>/lib/jaxp.properties

The above properties can be specified in jaxp.properties to define the behavior for all applications that use this Java Runtime. The format is "property-name=[value][,value]"*, for example:

javax.xml.accessExternalDTD=file,http

The property-names are the same as those of the System Properties that are: javax.xml.accessExternalDTD, javax.xml.accessExternalSchema, and javax.xml.accessExternalStylesheet.

4. Values of the proposed properties

All of the proposed properties above have values in the same format.

**Value:** a list of protocols separated by comma. A protocol is the scheme portion of an URI, or in the case of the JAR protocol, "jar" plus the scheme portion separated by colon. A scheme is defined as:

```
scheme = alpha * ( alpha | digit | "-" | "." )
```

where alpha = a-z and A-Z.

And the JAR protocol:

```
jar[:scheme]
```

Protocols are case-insensitive. Any whitespaces as defined by Character.isSpaceChar in the value will be ignored. Examples of protocols are file, http, jar:file.

**Default value:** the default value is implementation specific and therefore not specified. The following options are provided for consideration:

- an empty string to deny all access to external references;
- a specific protocol, such as file, to give permission to only the protocol;
- the keyword “all” to grant permission to all protocols.

When FEATURE_SECURE_PROCESSING is enabled, it is recommended that implementations restrict external connections by default, though this may cause problems for applications that process XML/XSD/XSL with external references.

**Granting all access:** the keyword "all" grants permission to all protocols. For example,
setting javax.xml.accessExternalDTD=all in jaxp.properties would allow a system to work as before with no restrictions on accessing external DTDs and Entity References.

5. Setting JAXP properties and features

JAXP properties can be set through JAXP factories as follows:

```java
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setAttribute(name, value);

SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser parser = spf.newSAXParser();
parser.setProperty(name, value);

XMLInputFactory xif = XMLInputFactory.newInstance();
xif.setProperty(name, value);

SchemaFactory schemaFactory = SchemaFactory.newInstance(schemaLanguage);
schemaFactory.setProperty(name, value);

Schema schema = schemaFactory.newSchema();
Validator validator = schema.newValidator();
validator.setProperty(name, value);

TransformerFactory factory = TransformerFactory.newInstance();
factory.setAttribute(name, value);
```

6. Scope and order

javax.xml.XMLConstants#FEATURE_SECURE_PROCESSING is a required feature for XML processors including DOM, SAX, Schema Validation, XSLT and XPath. It is recommended that implementations associate security related features and properties with the feature. When the secure feature is set to true, it requires that implementations limit XML processing to conform to implementation limits. When it is false, it instructs the implementation to process XML without such restrictions. For the new properties introduced in JAXP 1.5, it is recommended that when the secure feature is set to true, implementations restrict external connections, and when it is false, allow full access.

Properties specified in the jaxp.properties have effect all invocations of the JDK or JRE, and will override their default values, or those that may have been set by FEATURE_SECURE_PROCESSING.

System properties, when set, will affect one invocation only, and will override the default settings or those set in jaxp.properties, or those that may have been set by
FEATURE_SECURE_PROCESSING.

JAXP properties specified through JAXP factories or SAXParser will take preference over system properties, the jaxp.properties file, as well as javax.xml.XMLConstants#FEATURE_SECURE_PROCESSING.

The new JAXP properties shall have no effect on the relevant constructs they attempt to restrict in the following situations:

a) When there is a resolver and the source returned by the resolver is not null. This applies to entity resolvers that may be set on SAX and DOM parsers, XML resolvers on StAX parsers, LSResourceResolver on SchemaFactory, a Validator or ValidatorHandler, or URIResolver on a transformer.

b) When a schema is created explicitly by calling SchemaFactory's newSchema method

c) When external resources are not required. For example, the following features/properties are supported by the reference implementation and may be used to instruct the processor to not load the external DTD or resolve external entities.

- http://apache.org/xml/features/disallow-doctype-decl true
- http://apache.org/xml/features/nonvalidating/load-external-dtd false
- http://xml.org/sax/features/external-general-entities false
- http://xml.org/sax/features/external-parameter-entities false

7 Relationship with the Security Manager of the Java platform

a) The JAXP properties will be checked first before a connection is attempted whether or not a Security Manager is present. This means that a connection may be blocked even if it is granted permission by the Security Manager. For example, if the JAXP properties are set to disallow http protocol, they will effectively block any connection attempt even when an application has SocketPermission.

b) For the purpose of restricting connections, Security Manager is in a lower layer. Permissions will be checked down the process after the JAXP properties are evaluated. If an application does not have SocketPermission for example, it will receive a SecurityException even if the JAXP properties are set to allow http connection.

c) When Security Manager is present, the JAXP FEATURE_SECURE_PROCESSING is set to true. It is recommended that implementations set the values of the new JAXP properties as described in Item 4. Values and Item 6. Scope and order.

8. Error handling

If access to external resources is denied due to restrictions specified by the above access
properties, an exception will be thrown in accordance with that specified by the relevant processor as listed below.

**a) Exceptions**

org.xml.sax.SAXException in the process of parsing an XML file with
javax.xml.parsers.SAXParser, javax.xml.parsers.DocumentBuilder, and
javax.xml.stream.XMLInputFactory.

org.xml.sax.SAXException while creating a javax.xml.validation.Schema through
javax.xml.validation.SchemaFactory if the Schema file contains external DTD, or reference
to external schema.

org.xml.sax.SAXException while validating an XML file using
javax.xml.validation.Validator if the XML file references a Schema through
schemaLocation attribute

javax.xml.transform.TransformerConfigurationException while creating new
javax.xml.transform.Transformer using javax.xml.transform.TransformerFactory

**b) Error message format**

Implementations may consider error messages in the following format to provide users identifiable hint on why an error has been reported.

When access to external DTD is denied:
External DTD: Failed to read external DTD '<filename>', because '<protocol>' access is not allowed.

When access to external entity is denied:
External Entity: Failed to read external document '<filename>', because '<protocol>' access is not allowed.

When access to external Schema is denied:
schema_reference: Failed to read schema document '<filename>', because '<protocol>' access is not allowed.

When access to external Stylesheet is denied:
Could not read stylesheet target '<filename>', because '<protocol>' access is not allowed.

In all of the above error messages:
<filename> is the name of the external resource without file path;
<protocol> is the protocol denied, such as ‘file’.

**10.3 API Changes**

1. Changes to javax.xml.XMLConstants
Members
Field FEATURE_SECURE_PROCESSING
static java.lang.String FEATURE_SECURE_PROCESSING

Feature for secure processing is a required feature for XML processors including DOM, SAX, Schema Validation, XSLT and XPath. It is recommended that implementations associate security related features and properties with the feature.

• true instructs the implementation to process XML securely. It is recommended that implementations set limits and restrictions on XML processors as mitigation against vulnerabilities related to XML processing.

• false instructs the implementation to process XML without any limits and restrictions. Applications will be vulnerable to potential attacks if they process untrusted XML sources.

Field ACCESS_EXTERNAL_DTD
static java.lang.String ACCESS_EXTERNAL_DTD

Access External DTD
Restrict access to external DTDs, external Entity References to the protocols specified. If access is denied due to the restriction of this property, an exception defined by the processor will be thrown.

Value: a list of protocols separated by comma. A protocol is the scheme portion of an URI, or in the case of the JAR protocol, "jar" plus the scheme portion separated by colon. A scheme is defined as:

scheme = alpha *( alpha | digit | "+" | "-" | "." )
where alpha = a-z and A-Z.

And the JAR protocol:

jar[ :scheme]

Protocols are case-insensitive. Any whitespaces as defined by Character.isSpaceChar in the value will be ignored. Examples of protocols are file, http, jar:file.

Default value: the default value is implementation specific and therefore not specified. The following options are provided for consideration:

-- an empty string to deny all access to external references;
-- a specific protocol, such as file, to give permission to only the protocol;
-- the keyword “all” to grant permission to all protocols.

When FEATURE_SECURE_PROCESSING is enabled, it is recommended that implementations restrict external connections by default, though this may cause
problems for applications that process XML/XSD/XSL with external references.

**Granting all access:** the keyword "all" grants permission to all protocols.

**System Property:** The value of this property can be set or overridden by system property javax.xml.accessExternalDTD.

**${JAVA_HOME}/lib/jaxp.properties:** This configuration file is in standard java.util.Properties format. If the file exists and property javax.xml.accessExternalDTD is specified, its value will be used to override that of the property.

**Field ACCESS_EXTERNAL_SCHEMA**
static java.lang.String ACCESS_EXTERNAL_SCHEMA

Access External Schema

Restrict access to the protocols specified for external reference set by the schemaLocation attribute, Import and Include element. If access is denied due to the restriction of this property, an exception defined by the processor will be thrown.

**Value:** a list of protocols separated by comma. A protocol is the scheme portion of an URI, or in the case of the JAR protocol, "jar" plus the scheme portion separated by colon. A scheme is defined as:

```
scheme = alpha *( alpha | digit | "+" | "," | "," )
where alpha = a-z and A-Z.
```

And the JAR protocol:

```
jar[:scheme]
```

Protocols are case-insensitive. Any whitespaces as defined by Character.isSpaceChar in the value will be ignored. Examples of protocols are file, http, jar:file.

**Default value:** the default value is implementation specific and therefore not specified. The following options are provided for consideration:

- an empty string to deny all access to external references;
- a specific protocol, such as file, to give permission to only the protocol;
- the keyword “all” to grant permission to all protocols.

When FEATURE_SECURE_PROCESSING is enabled, it is recommended that implementations restrict external connections by default, though this may cause problems for applications that process XML/XSD/XSL with external references.
Granting all access: the keyword "all" grants permission to all protocols.

System Property: The value of this property can be set or overridden by system property javax.xml.accessExternalSchema.

${JAVA_HOME}/lib/jaxp.properties: This configuration file is in standard java.util.Properties format. If the file exists and property javax.xml.accessExternalSchema is specified, its value will be used to override that of the property.

Field ACCESS_EXTERNAL_STYLESHEET

static java.lang.String ACCESS_EXTERNAL_STYLESHEET

Access External Stylesheet

Restrict access to the protocols specified for external references set by the stylesheet processing instruction, Import and Include element. If access is denied due to the restriction of this property, an exception defined by the processor will be thrown.

Value: a list of protocols separated by comma. A protocol is the scheme portion of an URI, or in the case of the JAR protocol, "jar" plus the scheme portion separated by colon. A scheme is defined as:

scheme = alpha *( alpha | digit | "+" | "-" | "." )
where alpha = a-z and A-Z.

And the JAR protocol:

jar[:scheme]

Protocols are case-insensitive. Any whitespaces as defined by Character.isSpaceChar in the value will be ignored. Examples of protocols are file, http, jar:file.

Default value: the default value is implementation specific and therefore not specified. The following options are provided for consideration:

- an empty string to deny all access to external references;
- a specific protocol, such as file, to give permission to only the protocol;
- the keyword “all” to grant permission to all protocols.

When FEATURE_SECURE_PROCESSING is enabled, it is recommended that implementations restrict external connections by default, though this may cause problems for applications that process XML/XSD/XSL with external references.

Granting all access: the keyword "all" grants permission to all protocols.
System Property: The value of this property can be set or overridden by system property
javax.xml.accessExternalStylesheet.

${JAVA_HOME}/lib/jaxp.properties: This configuration file is in standard
java.util.Properties format. If the file exists and property javax.xml.accessExternalStylesheet is
specified, its value will be used to override that of the property.

2. Changes to javax.xml.parsers.DocumentBuilder
   Method setAttribute(String, Object)
Add the following description:

All implementations that implement JAXP 1.5 or newer are required to support the
ACCESS_EXTERNAL_DTD and ACCESS_EXTERNAL_SCHEMA properties.

Setting the ACCESS_EXTERNAL_DTD property restricts the access to external DTDs,
external Entity References to the protocols specified by the property. If access is denied during
parsing due to the restriction of this property, org.xml.sax.SAXException will be thrown by the
parse methods defined by javax.xml.parsers.DocumentBuilder.

Setting the ACCESS_EXTERNAL_SCHEMA property restricts the access to external Schema
set by the schemaLocation attribute to the protocols specified by the property. If access is
denied during parsing due to the restriction of this property, org.xml.sax.SAXException will be
thrown by the parse methods defined by javax.xml.parsers.DocumentBuilder.

3. Changes to javax.xml.parsers.SAXParser
   Method setProperty(String, Object)
Add the following description:

All implementations that implement JAXP 1.5 or newer are required to support the
ACCESS_EXTERNAL_DTD and ACCESS_EXTERNAL_SCHEMA properties.

Setting the ACCESS_EXTERNAL_DTD property restricts the access to external DTDs,
external Entity References to the protocols specified by the property. If access is denied during
parsing due to the restriction of this property, org.xml.sax.SAXException will be thrown by the
parse methods defined by javax.xml.parsers.SAXParser.

Setting the ACCESS_EXTERNAL_SCHEMA property restricts the access to external Schema
set by the schemaLocation attribute to the protocols specified by the property. If access is
denied during parsing due to the restriction of this property, org.xml.sax.SAXException will be
thrown by the parse methods defined by javax.xml.parsers.SAXParser.
4. Changes to javax.xml.stream.XMLInputFactory
   Method setProperty(String, Object)
   Add the following description:

   All implementations that implement JAXP 1.5 or newer are required to support the
   ACCESS_EXTERNAL_DTD property.

   Access to external DTDs, external Entity References is restricted to the protocols specified by
   the property. If access is denied during parsing due to the restriction of this property,
   javax.xml.stream.XMLStreamException will be thrown by the
   javax.xml.stream.XMLStreamReader#next() or
   javax.xml.stream.XMLEventReader#nextEvent() method.

5. Changes to javax.xml.transform.TransformerFactory
   Method setAttribute(String, Object)
   Add the following description:

   All implementations that implement JAXP 1.5 or newer are required to support the
   ACCESS_EXTERNAL_DTD and ACCESS_EXTERNAL_STYLESHEET properties.

   Access to external DTDs in the source file is restricted to the protocols specified by the
   ACCESS_EXTERNAL_DTD property. If access is denied during transformation due to the
   restriction of this property, javax.xml.transform.TransformerException will be thrown by the
   javax.xml.transform.Transformer.transform(Source, Result) method.

   Access to external DTDs in the stylesheet is restricted to the protocols specified by the
   ACCESS_EXTERNAL_DTD property. If access is denied during the creation of a new
   transformer due to the restriction of this property,
   javax.xml.transform.TransformerConfigurationException will be thrown by the
   newTransformer(Source) method.

   Access to external reference set by the stylesheet processing instruction, Import and Include
   element is restricted to the protocols specified by the ACCESS_EXTERNAL_STYLESHEET
   property. If access is denied during the creation of a new transformer due to the restriction of
   this property, javax.xml.transform.TransformerConfigurationException will be thrown by the
   newTransformer(Source) method.

   Access to external document through XSLT document function is restricted to the protocols
   specified by the property. If access is denied during the transformation due to the restriction of
   this property, javax.xml.transform.TransformerException will be thrown by the
   javax.xml.transform.Transformer.transform(Source, Result) method.

6. Changes to javax.xml.validation.SchemaFactory
   Method setProperty(String, Object)
Add the following description:

All implementations that implement JAXP 1.5 or newer are required to support the
ACCESS_EXTERNAL_DTD and ACCESS_EXTERNAL_SCHEMA properties.

Access to external DTDs in Schema files is restricted to the protocols specified by the
ACCESS_EXTERNAL_DTD property. If access is denied during the creation of new Schema
due to the restriction of this property, org.xml.sax.SAXException will be thrown by the
newSchema(Source) or newSchema(File) or newSchema(URL) or or newSchema(Source[]) method.

Access to external DTDs in xml source files is restricted to the protocols specified by the
ACCESS_EXTERNAL_DTD property. If access is denied during validation due to the
restriction of this property, org.xml.sax.SAXException will be thrown by the
javax.xml.validation.Validator.validate(Source) or Validator.validate(Source, Result) method.

Access to external reference set by the schemaLocation attribute is restricted to the protocols
specified by the ACCESS_EXTERNAL_SCHEMA property. If access is denied during
validation due to the restriction of this property, org.xml.sax.SAXException will be thrown by the
javax.xml.validation.Validator.validate(Source) or Validator.validate(Source, Result) method.

Access to external reference set by the Import and Include element is restricted to the protocols
specified by the ACCESS_EXTERNAL_SCHEMA property. If access is denied during
validation due to the restriction of this property, org.xml.sax.SAXException will be thrown by the
newSchema(Source) or newSchema(File) or newSchema(URL) or
newSchema(Source[]) method.

7. Changes to javax.xml.validation.Validator
   Method setProperty(String, Object)
   Add the following description:

   All implementations that implement JAXP 1.5 or newer are required to support the
   ACCESS_EXTERNAL_DTD and ACCESS_EXTERNAL_SCHEMA properties.

   Access to external DTDs in source or Schema file is restricted to the protocols specified by the
   ACCESS_EXTERNAL_DTD property. If access is denied during validation due to the
   restriction of this property, org.xml.sax.SAXException will be thrown by the validate(Source)
or validate(Source, Result) method.

   Access to external reference set by the schemaLocation attribute is restricted to the protocols
   specified by the ACCESS_EXTERNAL_SCHEMA property. If access is denied during
   validation due to the restriction of this property, org.xml.sax.SAXException will be thrown by the
   validate(Source) or validate(Source, Result) method.
10.4 Compatibility

The JAXP 1.5 specification does not require implementations to restrict connections by default. However, for implementations that do choose to do so, the behavior will not be backward compatible. Applications that process XML/XSD/XSL with external references will fail.

The system properties corresponding to each new property will help mitigate any compatibility issue in that users may change the settings without code changes.

The use of ${JAVA_HOME}/lib/jaxp.properties is a further attempt to reduce the impact. Where appropriate, a configuration of the new properties for the entire JDK can be done using this file.

Applications that use resolvers to handle external references, or use existing features or properties to specify not to load external resources, will not be affected by this change.
11. Changes Since JAXP 1.5

JAXP defines a number of service provider interfaces to allow deployment with alternative parser implementations (service providers). Service providers are located by means of:

1) Use a system property named after the corresponding factory name;

2) Use the properties file "lib/jaxp.properties" in the JRE directory;

3) Read JAR service file, for example, META-INF/services/java.xml.datatype.DatatypeFactory;

4) Fall back to the system default implementation.

The goal of JAXP 1.6 is to specify consistently that JAXP use java.util.ServiceLoader to replace the 3rd step above, for locating service provider implementations.

Furthermore, ServiceLoader will be the tool for finding service providers in the modularized, next-generation Java platform. Replacing the process with ServiceLoader allows for future deployments that may not be JAR files with META-INF/services configuration files.

11.1 Use java.util.ServiceLoader

Replace the 3rd step with the service-provider loading facility. The followings are changes to Chapter 4. Pluggability Layer.

11.1.1 SAX Plugability

Defined in the description of the following class and method:
public abstract class SAXParserFactory
public static SAXParserFactory newInstance()

1) The 3rd step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for the classname in the file META-INF/services/java.xml.datatype.DatatypeFactory;</td>
<td>Use the service-provider loading facilities, defined by the java.util.ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current</td>
</tr>
</tbody>
</table>
2) The 4th step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform default SAXParserFactory instance.</td>
<td>Otherwise the system-default implementation is returned.</td>
</tr>
</tbody>
</table>

### 11.1.2 DOM Plugability

Defined in the description of the following class and method:

```java
public abstract class DocumentBuilderFactory

public static DocumentBuilderFactory newInstance()
```

1) The 3rd step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the Services API (as detailed in the JAR specification), if available, to determine theclassname. The Services API will look for a classname in the file META-INF/services/javax.xml.parsers.DocumentBuilderFactory in jars available to the runtime.</td>
<td>Uses the service-provider loading facilities, defined by the java.util.ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.</td>
</tr>
</tbody>
</table>

2) The 4th step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform default <code>DocumentBuilderFactory</code> instance.</td>
<td>Otherwise, the system-default implementation is returned.</td>
</tr>
</tbody>
</table>
### 11.1.3 XSLT Plugability

Defined in the description of the following class and method:
public abstract class TransformerFactory
public static TransformerFactory newInstance() throws TransformerFactoryConfigurationException

1) The 3rd step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file META-INF/services/javax.xml.transform.TransformerFactory in jars available to the runtime.</td>
<td>Use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.</td>
</tr>
</tbody>
</table>

2) The 4th step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform default TransformerFactory instance.</td>
<td>Otherwise, the system-default implementation is returned.</td>
</tr>
</tbody>
</table>

### 11.1.4 XPath Plugability

Defined in the description of the following class and method:
public abstract class XPathFactory
public static final static XPathFactory newInstance()

1) The 3rd step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>The class loader is asked for service provider provider-configuration files matching javax.xml.xpath.XPathFactory in the resource directory META-INF/services. See the JAR File Specification for file format and parsing rules. Each potential service provider is required to implement the method: isObjectModelSupported(String</td>
<td>Use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.</td>
</tr>
</tbody>
</table>
The first service provider found in class loader order that supports the specified object model is returned. Implement the method isObjectModelSupported(String objectModel).

The first service provider found that supports the specified object model is returned. In case of ServiceConfigurationError an XPathFactoryConfigurationException will be thrown.

2) Other minor change:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static final XPathFactory newInstance()</td>
<td>public static XPathFactory newInstance()</td>
</tr>
</tbody>
</table>

11.1.5 Validation Plugability

Defined in the description of the following class and method:
public abstract class SchemaFactory
public static final static SchemaFactory newInstance(java.lang.String schemaLanguage)

1) The 3rd step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>The class loader is asked for service provider provider-configuration files matching javax.xml.validation.SchemaFactory in the resource directory META-INF/services. See the JAR File Specification for file format and parsing rules. Each potential service provider is required to implement the method: isSchemaLanguageSupported(String schemaLanguage)</td>
<td>Use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used. Each potential service provider is required to implement the method isSchemaLanguageSupported(String schemaLanguage) .</td>
</tr>
<tr>
<td>The first service provider found in class loader order that supports the specified schema language is returned.</td>
<td>The first service provider found that supports the specified schema language is returned.</td>
</tr>
</tbody>
</table>
In case of ServiceConfigurationError a SchemaFactoryConfigurationError will be thrown.

2) Other change:
Fix a typo in the description of class SchemaFactory, Schema Language section

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>implementors</td>
<td>implementors</td>
</tr>
</tbody>
</table>

3) New class

**public class SchemaFactoryConfigurationError**
Thrown when a problem with configuration with the Schema Factories exists. This error will typically be thrown when the class of a schema factory specified in the system properties cannot be found or instantiated.

**Synopsis:**
public SchemaFactoryConfigurationError extends Error {
    public SchemaFactoryConfigurationError();
    public SchemaFactoryConfigurationError(java.lang.String message);
    public SchemaFactoryConfigurationError(java.lang.String message, java.lang.Throwable cause);
    public SchemaFactoryConfigurationError(java.lang.Throwable cause);
}

**Inheritance Path:**
java.lang.Object
  • java.lang.Throwable
    • java.lang.Error
      • javax.xml.validation.SchemaFactoryConfigurationError

**Constructor Summary**

**Constructor and Description**

**SchemaFactoryConfigurationError ()**
Create a new SchemaFactoryConfigurationError with no detail message.

**SchemaFactoryConfigurationError (java.lang.String message)**
Create a new SchemaFactoryConfigurationError with the String specified as an error message.
**SchemaFactoryConfigurationError** (java.lang.String message, java.lang.Throwable cause)
Create a new SchemaFactoryConfigurationError with the given Throwable base cause and detail message.

**SchemaFactoryConfigurationError** (java.lang.Throwable cause)
Create a new SchemaFactoryConfigurationError with the given Throwable base cause.

### 11.1.6 Streaming API for XML Plugability

Defined in the description of the following classes and methods:
- public abstract class XMLEventFactory
- public static XMLEventFactory newInstance() throws FactoryConfigurationError
- public abstract class XMLInputFactory
- public static XMLInputFactory newInstance() throws FactoryConfigurationError
- public abstract class XMLOutputFactory
- public static XMLOutputFactory newInstance() throws FactoryConfigurationError

1) The 3rd step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file META-INF/services/javax.xml.stream.XMLEventFactory in jars available to the runtime.</td>
<td>Use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.</td>
</tr>
</tbody>
</table>

2) The 4th step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform default XMLEventFactory instance.</td>
<td>Otherwise, the system-default implementation is returned.</td>
</tr>
</tbody>
</table>
3) Error:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throws:</td>
<td>Throws:</td>
</tr>
<tr>
<td>FactoryConfigurationException - if an instance of this factory cannot be loaded</td>
<td>FactoryConfigurationException - in case of service configuration error or if the implementation is not available or cannot be instantiated.</td>
</tr>
</tbody>
</table>

### 11.1.7 Datatype Plugability

Defined in the description of the following class:

```java
public abstract class DatatypeFactory
```

1) The general statement:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>#newInstance() is used to create a new DatatypeFactory. The following implementation resolution mechanisms are used in the following order:</td>
<td>A new instance of the DatatypeFactory is created through the #newInstance() method that uses the following implementation resolution mechanisms to determine an implementation:</td>
</tr>
</tbody>
</table>

2) The 3rd step:

<table>
<thead>
<tr>
<th>Original Statement</th>
<th>New Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>The services resolution mechanism is used, e.g. META-INF/services/java.xml.datatype.DatatypeFactory. Any Exception thrown during the instantiation process is wrapped as a DatatypeConfigurationException.</td>
<td>Uses the service-provider loading facilities, defined by the java.util.ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used. In case of service configuration error a javax.xml.datatype.DatatypeConfigurationException will be thrown.</td>
</tr>
</tbody>
</table>
11.2 StAX 1.2, JSR 173 Stream API for XML MR3

The revision 1.2 of Stream API for XML Processing for the Java Platform, that is, JSR 173 Maintenance Review 3, deprecated newInstance methods in the StAX factories and added newFactory methods. The followings are API changes of StAX 1.2 with the ServiceLoader changes above incorporated in the description of the newFactory methods. Refer to Change Log for JSR-000173 Streaming API for XML, Maintenance Review 3.

11.2.1 Deprecated

Class:
javax.xml.stream.XMLEventFactory

Method:
public static XMLEventFactory newInstance(java.lang.String factoryId,
java.lang.ClassLoader classLoader throws FactoryConfigurationError

Add the following deprecation notice:
@deprecated to maintain API consistency. All newInstance methods are replaced with corresponding newFactory methods. The replacement newFactory(String factoryId, ClassLoader classLoader) method defines no changes in behavior from this method.

Class:
javax.xml.stream.XMLInputFactory

Method:
public static XMLInputFactory newInstance(java.lang.String factoryId,
java.lang.ClassLoader classLoader throws FactoryConfigurationError

Add the following deprecation notice:
@deprecated to maintain API consistency. All newInstance methods are replaced with corresponding newFactory methods. The replacement newFactory(String factoryId, ClassLoader classLoader) method defines no changes in behavior from this method.

Class:
javax.xml.stream.XMLOutputFactory

Method:
public static XMLOutputFactory newInstance(java.lang.String factoryId,
java.lang.ClassLoader classLoader throws FactoryConfigurationError

Add the following deprecation notice:
@deprecated This method has been deprecated because it returns an instance of XMLInputFactory, which is of the wrong class. Use the new method
newFactory(java.lang.String factoryId, java.lang.ClassLoader classLoader) instead.

11.2.2 New factory methods

11.2.2.1 javax.xml.stream.XMLEventFactory

Method:

public static XMLEventFactory newFactory() throws FactoryConfigurationError

Create a new instance of the factory.

This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the XMLEventFactory implementation class to load:

- Use the javax.xml.stream.XMLEventFactory system property.
- Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the system property defined above.
- Use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.
- Otherwise, the system-default implementation is returned.

Once an application has obtained a reference to a XMLEventFactory it can use the factory to configure and obtain stream instances.

Note that this is a new method that replaces the deprecated newInstance() method. No changes in behavior are defined by this replacement method relative to the deprecated method.

Throws:

FactoryConfigurationException - in case of service configuration error or if the implementation is not available or cannot be instantiated.

public static XMLEventFactory newFactory(java.lang.String factoryId, java.lang.ClassLoader classLoader) throws FactoryConfigurationException

Create a new instance of the factory. If the classLoader argument is null, then the ContextClassLoader is used.

This method uses the following ordered lookup procedure to determine the XMLEventFactory
implementation class to load:

- Use the value of the system property identified by `factoryId`.
- Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard `java.util.Properties` format and contains the fully qualified name of the implementation class with the key being the given `factoryId`.
- If `factoryId` is "javax.xml.stream.XMLEventFactory", use the service-provider loading facilities, defined by the `ServiceLoader` class, to attempt to locate and load an implementation of the service using the specified `ClassLoader`. If the `ClassLoader` is null, the default loading mechanism will apply: That is, the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.
- Otherwise, throws a `FactoryConfigurationError`.

Note that this is a new method that replaces the deprecated `newInstance(String factoryId, ClassLoader classLoader)` method. No changes in behavior are defined by this replacement method relative to the deprecated method.

Parameters:
- `factoryId` - Name of the factory to find, same as a property name
- `classLoader` - `ClassLoader` to use

Returns:
- the factory implementation

Throws:
- `FactoryConfigurationError` - in case of service configuration error or if the implementation is not available or cannot be instantiated.
- `FactoryConfigurationError` - if an instance of this factory cannot be loaded

11.2.2.2 `javax.xml.stream.XMLInputFactory`

Method:
`public static XMLInputFactory newFactory() throws FactoryConfigurationError`

Create a new instance of the factory.

This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the `XMLInputFactory` implementation class to load:

- Use the `javax.xml.stream.XMLInputFactory` system property.
- Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard `java.util.Properties` format and contains the fully qualified name of the implementation class with the key being the system property defined above.
- Use the service-provider loading facilities, defined by the `ServiceLoader` class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader.
loader to attempt to load the service. If the context class loader is null, the system class loader
will be used.
• Otherwise, the system-default implementation is returned.

Once an application has obtained a reference to a XMLInputFactory it can use the factory to
configure and obtain stream instances.

Note that this is a new method that replaces the deprecated newInstance() method. No changes in
behavior are defined by this replacement method relative to the deprecated method.

Throws:
    FactoryConfigurationError - in case of service configuration error or if the
implementation is not available or cannot be instantiated.

public static XMLInputFactory newFactory(java.lang.String factoryId,
    java.lang.ClassLoader classLoader) throws FactoryConfigurationError

Create a new instance of the factory. If the classLoader argument is null, then the
ContextClassLoader is used.

This method uses the following ordered lookup procedure to determine the XMLInputFactory
implementation class to load:
• Use the value of the system property identified by factoryId.
• Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in
standard java.util.Properties format and contains the fully qualified name of the
implementation class with the key being the given factoryId.
• If factoryId is "javax.xml.stream.XMLInputFactory", use the service-provider loading
facilities, defined by the ServiceLoader class, to attempt to locate and load an
implementation of the service using the specified ClassLoader. If the classLoader is null, the
default loading mechanism will apply: That is, the service-provider loading facility will use
the current thread's context class loader to attempt to load the service. If the context class
loader is null, the system class loader will be used.
• Otherwise, throws a FactoryConfigurationException.

Note that this is a new method that replaces the deprecated
newInstance(String
    factoryId, ClassLoader classLoader) method. No changes in behavior are defined
by this replacement method relative to the deprecated method.

Parameters:
    factoryId - Name of the factory to find, same as a property name
    classLoader - classLoader to use

Returns:
    the factory implementation

 Throws:
FactoryConfigurationError - in case of service configuration error or if the implementation is not available or cannot be instantiated.
FactoryConfigurationError - if an instance of this factory cannot be loaded.

11.2.2.3 javax.xml.stream.XMLOutputFactory

Method:
public static XMLOutputFactory newFactory() throws FactoryConfigurationError

Create a new instance of the factory.

This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the XMLOutputFactory implementation class to load:

• Use the javax.xml.stream.XMLOutputFactory system property.
• Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the system property defined above.
• Use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.
• Otherwise, the system-default implementation is returned.

Once an application has obtained a reference to a XMLOutputFactory it can use the factory to configure and obtain stream instances.

Note that this is a new method that replaces the deprecated newInstance() method. No changes in behavior are defined by this replacement method relative to the deprecated method.

Throws:
FactoryConfigurationError - in case of service configuration error or if the implementation is not available or cannot be instantiated.

public static XMLOutputFactory newFactory(java.lang.String factoryId,
java.lang.ClassLoader classLoader) throws FactoryConfigurationError

Create a new instance of the factory. If the classLoader argument is null, then the ContextClassLoader is used.

This method uses the following ordered lookup procedure to determine the XMLOutputFactory
implementation class to load:

- Use the value of the system property identified by `factoryId`.
- Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the given `factoryId`.
- If `factoryId` is "javax.xml.stream.XMLOutputFactory", use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the specified ClassLoader. If the classLoader is null, the default loading mechanism will apply: That is, the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.
- Otherwise, throws a FactoryConfigurationError.

Note that this is a new method that replaces the deprecated newInstance(String _factoryId_, ClassLoader classLoader) method. No changes in behavior are defined by this replacement method relative to the deprecated method.

Parameters:
  factoryId - Name of the factory to find, same as a property name
  classLoader - classLoader to use

Returns:
  the factory implementation

Throws:
  FactoryConfigurationError - in case of service configuration error or if the implementation is not available or cannot be instantiated.
  FactoryConfigurationError - if an instance of this factory cannot be loaded

11.3 API package org.w3c.dom.views

This specification includes the following API package by reference:
  org.w3c.dom.views

The org.w3c.dom.views package includes the following interfaces:

```java
public interface AbstractView
    public DocumentView getDocument();

public interface DocumentView
    public AbstractView getDefaultView();
```

11.4 Compatibility for the ServiceLoader change

This specification mandates the use of java.util.ServiceLoader for finding service providers. Service providers across JAXP will now be located consistently following the process as defined in ServiceLoader. This change may represent some subtle difference from
implementations of previous versions of the specification where the provider-configuration file may have been located differently, for example, by using a different getXXX method of the ClassLoader than ServiceLoader. Applications that implement their own Classloaders shall therefore make sure that the ClassLoaders’ getXXX methods are implemented consistently so as to maintain compatibility.

The StAX API, JSR 173, defined newInstance/newFactory method with a factoryId as a parameter. Since there was no constraint on what the value could be in the StAX specification, it implied it could be any arbitrary string. With this specification change, in the context of JAXP, the value of factoryId must be the name of the base service class if it is intended to represent the name of the service configuration file, that is, if it is not the name of a System Property.

11.5 End of JSR 206 Java™API for XML Processing

Since JAXP version 1.1, JSR 206 has been distributed as a standalone technology and part of the Java SE at the same time. The JAXP standalone project has played its vital role as an open-source project. However, starting from OpenJDK 7, the JAXP source has been merged into the OpenJDK repository, and the development has been conducted within the OpenJDK family. There is therefore no need to maintain a separate open-source project in JAXP.

In accordance with JCP 2.9 Process Document, item 2.1.4 Platform Inclusion, it is announced that the JAXP Standalone distribution will end after MR3, JAXP 1.6. The technology that JSR 206 defines will be delivered as a part of the Java SE solely. Future changes in the JAXP API will be defined through the Platform JSR. JAXP will no longer exist as a standalone library.