Java API for XML Processing

Version 1.1 Public Review

Comments to: jsr63-comments@eng.sun.com

James Duncan Davidson

Rajiv Mordani

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto CA 94303 USA
650 960-1300

October 2, 2000

Java(TM) API for XML Processing (JAXP) Specification ("Specification")
Version: 1.1
Status: Pre-FCS
Release: September 28, 2000

OR OTHER RIGHTS. This document does not represent any commitment to release or implement any portion of the Specification in any product.

THE SPECIFICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. SUN MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

LIMITATION OF LIABILITY
TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN 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 TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF SUN AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend Sun and its licensors from any claims based on your use of the Specification for any purposes other than those of internal evaluation, and from 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
If this Software 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
You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your evaluation of the Specification ("Feedback"). To the extent that you provide Sun with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Sun 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 related to the Specification and future versions, implementations, and test suites thereof.

SECTION 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 are a partnership made in heaven. 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 1.1. 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 2.0 (JCP 2.0).

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.

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

1.5 Development of this Specification

This specification was developed in accordance with the Java Community Process 2.0. It was developed under the authorization of Java Specification Request 63. More information about the Java Community Process can be found at:

`http://java.sun.com/jcp/`

The specific information contained in Java Specification Request 63 can be found at:

`http://java.sun.com/aboutJava/communityprocess/jsr/jsr_063_jaxp.html`

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

- James Duncan Davidson (co-lead), Sun Microsystems
- Rajiv Mordani (co-lead), Sun Microsystems
- Jeff Mischinkinsky, Persistence
- Todd Karakashain, BEA
- Tom Reilly, Allaire
- Tom Bates, Informix
- Miles Sabin, CromwellMedia
- Wolfram Kaiser, POET
- Paul Boutros, eBusiness Technologies
- Pier Fumagalli, Apache Software Foundation
- Stefano Mazzocchi, Apache Software Foundation
- Takuki Kamiya, Fujitsu Ltd

1.6 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:

`jsr63-comments@eng.sun.com`

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 located at:

`http://java.sun.com/xml/`

1.7 Acknowledgements

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

- David Megginson and the XML-DEV community who developed the SAX API
- The W3C DOM Working Group chaired by Lauren Wood
- The JSR-63 Expert Group listed above
- Graham Hamilton, Mark Hapner, Eduardo Pelegri-Lopart, Connie Weiss, Jim Driscoll, Edwin Goei, Costin Manolache, Mark Reinhold, Bill Shannon and Will Iverson all of whom work at Sun Microsystems and whose talents have all reflected upon the development of this API.

SECTION 2

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.

2.1 W3C XML 1.0 Recommendation

The W3C XML 1.0 Recommendation specifies the core XML syntax by subsetting the existing, widely used international SGML[1] text processing standard. It is a product of the W3C XML Activity, details of which can be found at:

`http://www.w3.org/XML/`

The XML 1.0 Recommendation can is located at:

`http://www.w3.org/TR/1998/REC-xml-19980210`

1. Standard Generalized Markup Language, ISO 8879:1986(E) as amended and corrected.

This specification includes by reference the XML 1.0 Recommendation in its entirety for the purposes of defining the XML language manipulated by the APIs defined herein.

2.2 W3C XML Namespaces 1.0 Recommendation

The W3C XML Namespaces Recommendation defines the syntax and semantics for XML structures required to be distinct from other XML markup. In particular, it defines a mechanism whereby a set of XML markup may have a distinguishing "namespace" associated with it, and the responsibility of XML parser in handling and exposing such namespace information.

The XML Namespaces 1.0 Recommendation is located at:

`http://www.w3.org/TR/1999/REC-xml-names-19990114/`

This specification includes by reference the XML Namespaces 1.0 Recommendation in its entirety.

2.3 Simple API for XML Parsing (SAX) 2.0

The Simple API for XML (SAX) is a public domain API developed cooperatively by the members of the XML-DEV mailing list. It provides an event-driven interface to the process of parsing an XML document.

An event driven interface provides a mechanism for a "callback" notifications to application’s code as the underlying parser recognizes XML syntactic constructions in the document.

The SAX 2.0 API is located at:

`http://www.megginson.com/SAX/index.html`

The pre 1.0 version of SAX 2 extensions is located at:

`http://www.megginson.com/Software/sax2-ext-1.0pre.zip`

The details of the XML-DEV mailing list can be found at

http://xml.org/xml-dev/index.shtml

As of this writing SAX2 extensions is in pre 1.0 stage. This specification includes by reference the SAX 2.0 API and the pre 1.0 version of SAX2 extensions in its entirety.

The API packages included by reference are:

- org.xml.sax
- org.xml.sax.helpers
- org.xml.sax.ext

2.4 Document Object Model (DOM) Level 2

The Document Object Model (DOM) is a set of interfaces defined by the W3C DOM Working Group. It describes facilities for a programmatic representation of a parsed XML (or HTML) document. The DOM Level 2 specification defines these interfaces using Interface Definition Language (IDL) in a language independent fashion and also includes a Java Language binding.

The DOM Level 2 Core Proposed Recommendation is located at:

http://www.w3.org/TR/2000/PR-DOM-Level-2-Core-20000927/

As of this writing the DOM Level 2 Core specification is still in Proposed Recommendation. This specification includes by reference both the abstract semantics described for the DOM Level 2 Core Proposed Recommendation interfaces and the associated Java Language binding. It does not include the optional extensions defined by the DOM working group. The API package included by this specification is:

- org.w3c.dom

2.5 XSLT 1.0

The XSL Transformations (XSLT) describes a language for transforming XML documents into other XML documents or other text output. It was defined by the W3C XSL Working group.

The XSLT 1.0 Recommendation is located at:

```
http://www.w3.org/TR/1999/REC-xslt-19991116
```

This specification includes by reference the XSLT 1.0 specification in its entirety.

SECTION 3

Plugability Layer

The endorsed APIs provide broad and useful functionality. However, the use of a SAX or a DOM parser typically requires knowledge of the specific implementation of the parser. 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 a compliant SAX or DOM parser to be used through the abstract `javax.xml.parsers` API.

3.1 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 the static `newInstance` method 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 JAVA_HOME (the parent directory where jdk is installed)/lib/jaxp.properties for a property file that contains the name of the implementation class keyed on the same value as the system property defined above.
- 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 FactoryConfigurationError 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.

3.1.1 Examples

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

```
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:

```
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 shown below

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

3.2 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 the static `newInstance` method 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 JAVA_HOME (the parent directory where jdk is installed)/lib/jaxp.properties for a property file that contains the name of the implementation class keyed on the same value as the system property defined above.
- 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.DocumentBuilderFactory` in jars available to the runtime.
- Platform default `DocumentBuilderFactory` instance.

If the `DocumentBuilderFactory` implementation class cannot be loaded or instantiated at runtime, a `FactoryConfigurationError` 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.

3.2.1 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.

3.2.2 Examples

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

```
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:

```
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 shown below

```
java -Djavax.xml.parsers.DocumentBuilderFac-
tory=org.apache.xerces.jaxp.DocumentBuilderFactoryImpl
user.parserApp.
```

3.3 XSLT Plugability

The XSLT Plugability classes allow an application programmer to obtain a `Transform` object that is based on a specific XSLT stylesheet from a `TransformFactory` implementation. In order to obtain a `Transform` object, a pro-

grammer first obtains an instance of the `TransformFactory`. The `TransformFactory` instance is obtained via the static newInstance method of the `TransformFactory` class.

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

- Use the `javax.xml.parsers.TransformFactory` system property
- Use the JAVA_HOME (the parent directory where jdk is installed)/lib/jaxp.properties for a property file that contains the name of the implementation class keyed on the same value as the system property defined above.
- 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.TransformFactory` in jars available to the runtime.
- Platform default `TransformFactory` instance.

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

3.3.1 Examples

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

```
Transform transform;
TransformFactory factory = TransformFactory.newInstance();
String stylesheet = "/home/user/mystylesheet.xsl";
try {
    transform = factory.newTransform(new File(stylesheet));
    File in = new File("/home/user/sourcefile.xml");
    File out = new File("/home/user/destfile.html");
    transform.transform(in, out);
} catch (IOException ioe) {
    // handle error
}
```

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

```
java -Djavax.xml.transform.TransformFac-
tory=org.apache.xerces.jaxp.TransformFactoryImpl
user.parserApp.
```

3.4 Thread Safety

Implementations of the `SAXParser`, `DocumentBuilder` and `Transform` abstract classes are not expected to be thread safe by this specification. This means that application programmers should not expect to be able to use the same instance of a `SAXParser`, `DocumentBuilder` or `Transform` 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 given `SAXParser`, `DocumentBuilder` or `Transform` instance.

Configuration of a `SAXParserFactory`, `DocumentBuilderFactory or TransformFactory` is also not expected to be thread safe. This means that an application programmer should not allow a `SAXParserFactory` or `DocumentBuilderFactory` to have its `setNamespaceAware` or `setValidating` methods from more than one thread.

It is expected that the `newSAXParser` method of a `SAXParserFactory` implementation, the `newDocumentBuilder` method of a `DocumentBuilderFactory` and the `newTransform` method of a `TransformFactory` will be thread safe without side effects. This means that an application programmer should expect to be able to create parser instances in multiple threads at once from a shared factory without side effects or problems.

SECTION 4

Packages javax.xml.parsers and javax.xml.transform

This section defines the API of the `javax.xml.parsers and javax.xml.transform` packages.

4.1 public abstract class SAXParserFactory

The `SAXParserFactory` defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents.

```
public abstract class SAXParserFactory {
    protected SAXParserFactory();
    public static SAXParserFactory newInstace();
    public abstract SAXParser newSAXParser()
        throws ParserConfigurationException, SAXException;
    public void setNamespaceAware(boolean aware);
    public void setValidating(boolean validating);
    public boolean isNamespaceAware();
    public boolean isValidating();
        public abstract void setFeature(String name,
                                        boolean value) throws
          SAXNotRecognizedException, SAXNotSupportedException
    public abstract boolean getFeature (String name) throws
          SAXNotRecognizedException, SAXNotSupportedException
```

```
}
```

4.1.1 protected SAXParserFactory()

An empty constructor is provided. Implementors of this abstract class must provide their own public no-argument constructor in order for the static `newInstance` method to work correctly. Application programmers should be able to instantiate an implementation of this abstract class directly if they want to use a specific implementation of this API without using the static newInstance method to obtain the configured or platform default implementation.

4.1.2 public static SAXParserFactory newInstance()

Returns a new instance of a `SAXParserFactory`. Every call to `newInstance` will return a unique instance of `SAXParserFactory`.

Throws a `FactoryConfigurationError` if the class implementing the factory cannot be found or instantiated. An Error is thrown instead of an exception because the application is not expected to handle or recover from such events.

4.1.3 public abstract SAXParser newSAXParser()

Returns a new configured instance of type `SAXParser`.

Throws a `ParserConfigurationException` if the `SAXParser` instance cannot be created with the requested configuration.

Implementation of the `SAXParser` class is not affected by subsequent changes in their factories configuration.

Throws a `SAXException` if the initialization of the underlying parser fails.

4.1.4 public void setNamespaceAware(boolean aware)

Configuration method that specifies whether the parsers created by this factory are required to provide XML namespace support or not.

Note, if a parser cannot be created by this factory that satisfies the requested namespace awareness value, a `ParserConfigurationException` will be thrown when the program attempts to acquire the parser via the `newSaxParser` method.

4.1.5 public void setValidating(boolean validating)

Configuration method whether specifies if the parsers created by this factory are required to validate the XML documents that they parse.

Note, that if a parser cannot be created by this factory that satisfies the requested validation capacity, a `ParserConfigurationException` will be thrown when the application attempts to acquire the parser via the `newSaxParser` method.

4.1.6 public boolean isNamespaceAware()

Indicates if this `SAXParserFactory` is configured to produce parsers that are namespace aware or not.

4.1.7 public boolean isValidating()

Indicates if this `SAXParserFactory` is configured to produce parsers that validate XML documents as they are parsed.

4.1.8 public abstract void setFeature(String name, boolean value)

Configuration mechanism that specifies that the parsers created by this factory are configured with the given feature.This API was introduced in SAX 2.0 and allow users to configure the underlying SAX parser.

Throws a `SAXNotRecognizedException` if the underlying parser can't recognize the option.

Throws a `SAXNotSupportedException` if the underlying parser recognizes but doesn't support the option.

4.1.9 public abstract boolean getFeature(String name)

Returns the value of the requested feature.

Throws a SAXNotRecognizedException if the underlying parser can't recognize the option.

Throws a SAXNotSupportedException if the underlying parser recognizes but doesn't support the option.

4.2 public abstract class SAXParser

Implementation instances of the SAXParser abstract class contain an implementation of the org.xml.saxParser interface and enables content from a variety of sources to be parsed using the contained parser. Instances of SAXParser are obtained from a SAXParserFactory by invoking its newSAXParser method.

```
public abstract class SAXParser {
    protected SAXParser();
    public abstract void setProperty(String name,
                                     Object value) throws
          SAXNotRecognizedException, SAXNotSupportedException
    public abstract Object getProperty (String name) throws
          SAXNotRecognizedException, SAXNotSupportedException

    public void parse(InputStream stream, HandlerBase base)
        throws SAXException, IOException;
    public void parse(InputStream stream, HandlerBase base,
                      String systemId)
        throws SAXException, IOException;
    public void parse(String uri, HandlerBase base)
        throws SAXException, IOException;
    public void parse(File file, HandlerBase base)
        throws SAXException, IOException;
    public void parse(InputSource source, HandlerBase base)
        throws SAXException, IOException;
    public void parse(InputStream stream, DefaultHandler dh)
        throws SAXException, IOException;
public void parse(InputStream stream, DefaultHandler dh,
                  String systemId)
        throws SAXException, IOException;
    public void parse(String uri, DefaultHandler dh)
        throws SAXException, IOException;
    public void parse(File file, DefaultHandler dh)
        throws SAXException, IOException;
    public void parse(InputSource source, DefaultHandler dh)
```

```
        throws SAXException, IOException;
    public abstract org.xml.sax.Parser getParser();
        throws SAXException;
    public abstract org.xml.sax.XMLReader getXMLReader();
        throws SAXException;
    public abstract boolean isNamespaceAware();
    public abstract boolean isValidating();
}
```

4.2.1 protected SAXParser()

An empty constructor is provided. Implementations should provide a protected constructor so that their factory implementation can instantiate instances of the implementation class. Application programmers should not be able to directly construct implementation subclasses of this abstract subclass. The only way a application should be able to obtain a reference to a `SAXParser` implementation instance is by using the appropriate methods of the `SAXParserFactory`.

4.2.2 public abstract void setProperty(String name, Object value)

Allows implementation specific properties to be set. This API was introduced in SAX 2.0 and to allow users to configure the underlying SAX parser.

Throws a `SAXNotRecognizedException` if the underlying parser can't recognize the option.

Throws a `SAXNotSupportedException` if the underlying parser recognizes but doesn't support the option.

4.2.3 public abstract Object getProperty(String name)

Returns the value of the requested property.

Throws a `SAXNotRecognizedException` if the underlying parser can't recognize the option.

Throws a `SAXNotSupportedException` if the underlying parser recognizes but doesn't support the option.

4.2.4 public void parse(InputStream stream, HandlerBase base)

Parses the contents of the given `java.io.InputStream` as an XML document using the specified `org.xml.sax.HandlerBase` object. *Use of the DefaultHandler version of this method is recommended.*

Throws an `org.xml.sax.SAXException` if there is a problem parsing the given XML content.

Throws a `java.io.IOException` if any IO errors occur reading the given `InputStream`.

Throws an `IllegalArgumentException` if the given `InputStream` is null.

4.2.5 public void parse(InputStream stream, HandlerBase base, String systemId)

Parses the contents of the given `java.io.InputStream` as an XML document using the specified `org.xml.sax.HandlerBase` object. The systemId provides a base for resolving relative URIs. *Use of the DefaultHandler version of this method is recommended.*

Throws an `org.xml.sax.SAXException` if there is a problem parsing the given XML content.

Throws a `java.io.IOException` if any IO errors occur reading the given `InputStream`.

Throws an `IllegalArgumentException` if the given `InputStream` is null.

4.2.6 public void parse(String uri, HandlerBase base)

Parses the content of the given URI as an XML document using the specified `org.sax.xml.HandlerBase` object. *Use of the DefaultHandler version of this method is recommended.*

Throws an `org.xml.sax.SAXException` if there is a problem parsing the given XML content.Throws a `java.io.IOException` if any IO errors occur while reading content located by the given URI.

Throws an `IllegalArgumentException` if the given URI is null.

4.2.7 public void parse(File file, HandlerBase base)

Parses the content of the given java.io.File as an XML document using the specified `org.sax.xml.HandlerBase` object. *Use of the DefaultHandler version of this method is recommended.*

Throws an `org.xml.sax.SAXException` if there is a problem parsing the given XML content.

Throws a `java.io.IOException` if any IO errors occur while reading content from the given `File`.

Throws an `IllegalArgumentException` if the given `File` is null.

4.2.8 public void parse(InputSource source, HandlerBase base)

Parses the content of the given `org.xml.sax.InputSource` as an XML document using the specified `org.sax.xml.HandlerBase` object. *Use of the DefaultHandler version of this method is recommended.*.

Throws an `org.xml.sax.SAXException` if there is a problem parsing the given XML content.

Throws a `java.io.IOException` if any IO Errors occur while reading content from the given InputSource.

Throws an `IllegalArgumentException` if the given InputSource is null.

4.2.9 public void parse(InputStream stream, DefaultHandler dh)

Parses the contents of the given `java.io.InputStream` as an XML document using the specified `org.xml.sax.DefaultHandler` object.

Throws an `org.xml.sax.SAXException` if there is a problem parsing the given XML content.

Throws a `java.io.IOException` if any IO errors occur reading the given `InputStream`.

Throws an `IllegalArgumentException` if the given `InputStream` is null.

4.2.10 public void parse(InputStream stream, DefaultHandler dh, String systemId)

Parses the contents of the given `java.io.InputStream` as an XML document using the specified `org.xml.sax.DefaultHandler` object. The systemId provides a base for resolving relative URIs.

Throws an `org.xml.sax.SAXException` if there is a problem parsing the given XML content.

Throws a `java.io.IOException` if any IO errors occur reading the given `InputStream`.

Throws an `IllegalArgumentException` if the given `InputStream` is null.

4.2.11 public void parse(String uri, DefaultHandler dh)

Parses the content of the given URI as an XML document using the specified `org.sax.xml.DefaultHandler` object.

Throws an `org.xml.sax.SAXException` if there is a problem parsing the given XML content.Throws a `java.io.IOException` if any IO errors occur while reading content located by the given URI.

Throws an `IllegalArgumentException` if the given URI is null.

4.2.12 public void parse(File file, DefaultHandler dh)

Parses the content of the given java.io.File as an XML document using the specified `org.sax.xml.DefaultHandler` object.

Throws an `org.xml.sax.SAXException` if there is a problem parsing the given XML content.

Throws a `java.io.IOException` if any IO errors occur while reading content from the given `File`.

Throws an `IllegalArgumentException` if the given `File` is null.

4.2.13 public void parse(InputSource source, DefaultHandler dh)

Parses the content of the given `org.xml.sax.InputSource` as an XML document using the specified `org.sax.xml.DefaultHandler` object.

Throws an `org.xml.sax.SAXException` if there is a problem parsing the given XML content.

Throws a `java.io.IOException` if any IO Errors occur while reading content from the given InputSource.

Throws an `IllegalArgumentException` if the given InputSource is null.

4.2.14 public abstract org.xml.sax.Parser getParser()

Returns the underlying `org.xml.sax.Parser` object which is wrapped by this `SAXParser` implementation. *Use of the getXMLReader method is recommended.*

Throws a `SAXException` if the underlying parser cannot be obtained.

4.2.15 public abstract boolean isNamespaceAware()

Returns whether or not this parser supports XML namespaces.

4.2.16 public abstract boolean isValidating()

Returns whether or not this parser supports validating XML content as it is parsed.

4.3 public abstract class DocumentBuilderFactory

The `DocumentBuilderFactory` defines a factory API that enables applications to configure and obtain a parser to parse XML documents into a DOM `Document` tree.

```
public abstract class DocumentBuilderFactory {
    protected DocumentBuilderFactory();
    public static DocumentBuilderFactory newInstance();
    public DocumentBuilder newDocumentBuilder()
        throws ParserConfigurationException;
    public void setNamespaceAware(boolean awareness);
    public void setValidating(boolean validating);
    public void setIgnoreElementContentWhitespace(boolean
whitespace);
    public void setExpandEntityReferences(boolean
                                          expandEntityRef);
    public void setIgnoringComments(boolean ignoreComments);
    public void setCoalescing(boolean coalescing);
    public boolean isNamespaceAware();
    public boolean isValidating();
    public boolean isIgnoreElementContentWhitespace();
    public boolean isExpandEntityReferences();
    public boolean isIgnoringComments();
    public boolean isCoalescing();
    public abstract void setAttribute(String name,
                                      Object value)
                     throws IllegalArgumentException;
    public abstract Object getAttribute(String name)
                     throws IllegalArgumentException;
}
```

4.3.1 protected DocumentBuilderFactory()

An empty constructor is provided by the API. Implementors of this abstract class must provide a public no-argument constructor in order for the static `newInstance` method to work correctly. Application programmers should be able to instantiate an implementation of this abstract class directly if they want to use a specific implementation of this API without using the static newInstance method to obtain the configured or platform default implementation.

4.3.2 public static DocumentBuilderFactory newInstance()

Returns a new instance of a `DocumentBuilderFactory`. Every call to `newInstance` will return a unique instance of `DocumentBuilderFactory`.

Throws a `FactoryConfigurationError` if the class implementing the factory cannot be found or instantiated. An Error is thrown instead of an exception because the application is not expected to handle or recover from such events.

4.3.3 public DocumentBuilder newDocumentBuilder()

Returns a new configured instance of type `DocumentBuilder`.

Implementation of the `DocumentBuilder` class is not affected by subsequent changes in their factories configuration.

Throws a `ParserConfigurationException` if the `DocumentBuilder` instance cannot be created with the requested configuration.

4.3.4 public void setNamespaceAware(boolean aware)

Configuration method that specifies whether the parsers created by this factory are required to provide XML namespace support or not.

Note that if a parser cannot be created by this factory that satisfies the requested namespace awareness, a `ParserConfigurationException` will be thrown when an attempt to obtain the parser via the `newSaxParser` method is made.

4.3.5 public void setValidating(boolean validating)

Configuration method that specifies if the parsers created by this factory are required to validate the XML documents that they parse.

Note that if a parser cannot be created by this factory that satisfies the requested validation capacity, a `ParserConfigurationException`will be thrown when an attempt to obtain the parser via the `newSaxParser` method is made.

4.3.6 public void setIgnoreElementContentWhitespace(boolean whitespace)

Configuration method which specifies that parsers created by this factory must eliminate whitespace in element content (sometimes known loosely as 'ignorable whitespace') when parsing XML documents (see XML REC 2.10).

Note that only whitespace which is directly contained within an element that has an element only content model (see XML REC 3.2.1) will be eliminated. Due to reliance on the content model this setting requires the parser to be in validating mode.

4.3.7 public void setExpandEntityReferences(boolean expand)

Configuration method that specifies if the parsers created by this factory will expand entity references in the XML documents that they parse as defined in Appendix D of the XML recommendation

4.3.8 public void setIgnoringComments(boolean ignoreComments)

Configuration method that specifies if the parsers created by this factory are required to ignore comments in the XML documents that they parse.

4.3.9 public void setCoalescing(boolean coalescing)

Configuration method that specifies if the parsers created by this factory are required to convert CDATASections to a Text node and append it to the adjacent (if any) text node.

4.3.10 public boolean isNamespaceAware()

Indicates if this `DocumentBuilderFactory` is configured to produce parsers that are namespace aware or not.

4.3.11 public boolean isValidating()

Indicates if this `DocumentBuilderFactory` is configured to produce parsers that validate XML documents as they are parsed.

4.3.12 public boolean isIgnoreElementContentWhitespace()

Indicates if this `DocumentBuilderFactory` is configured to produce parsers that ignore "ignorable whitespace" in XML documents as they are parsed.

4.3.13 public boolean isExpandEntityReferences()

Indicates if this `DocumentBuilderFactory` is configured to produce parsers that expand entity references in XML documents as they are parsed.

4.3.14 public boolean isIgnoringComments()

Indicates if this `DocumentBuilderFactory` is configured to produce parsers that ignore comments in XML documents as they are parsed.

4.3.15 public boolean isCoalescing()

Indicates if this `DocumentBuilderFactory` is configured to produce parsers that convert CDATA nodes to Text nodes and append it to the adjacent (if any) Text node in XML documents as they are parsed.

4.3.16 public abstract void setAttribute(String name, Object value)

Allows the user to set specific attributes provided by the underlying implementation. It is recommended that the name used to describe these attributes be unique and one way to achieve this could be to use the package naming convention. The names beginning with java and javax are reserved by the specification

Throws `IllegalArgumentException` if the underlying implementation doesn’t recognize the attribute.

4.3.17 public abstract Object getAttribute(String name)

Returns the attribute value.

Throws `IllegalArgumentException` if the underlying implementation doesn’t recognize the attribute.

4.4 public abstract class DocumentBuilder

Instances of `DocumentBuilder` provide a mechansim for parsing XML documents into a DOM document tree represented by an `org.w3c.dom.Document` object. A `DocumentBuilder` instance is obtained from a `DocumentBuilderFactory` by invoking its `newDocumentBuilder` method.

Note that DocumentBuilder uses several classes from the SAX API. This does not require that the implementor of the underlying DOM implementation use a SAX parser to parse XML content into a `org.w3c.dom.Document`. It merely requires that the implementation communicate with the application using these existing APIs.

```
public abstract class DocumentBuilder {
    protected DocumentBuilder();
    public Document parse(InputStream is)
        throws SAXException, IOException;
    public Document parse(InputStream is, String systemId)
        throws SAXException, IOException;
    public Document parse(String uri)
        throws SAXException, IOException;
    public Document parse(File f)
        throws SAXException, IOException;
    public abstract Document parse(InputSource is)
        throws SAXException, IOException;
    public abstract boolean isNamespaceAware();
    public abstract boolean isValidating();
    public abstract void setEntityResolver(EntityResolver er);
    public abstract void setErrorHandler(ErrorHandler eh);
    public Document newDocument();
}
```

4.4.1 protected DocumentBuilder()

An empty constructor is provided. Implementations should provide a protected constructor so that their factory implementation can instantiate instances of the implementation class. Application programmers should not be able to directly construct implementation subclasses of this abstract subclass. The only way a application should be able to obtain a reference to a `DocumentBuilder` implementation instance is by using the appropriate methods of the `DocumentBuilder`.

4.4.2 public Document parse(InputStream stream)

Parses the contents of the given `java.io.InputStream` as an XML document and returns an `org.w3c.dom.Document` object.

Throws a `java.io.IOException` if any IO errors occur reading the given `InputStream`.

Throws an `IllegalArgumentException` if the given `InputStream` is null.

4.4.3 public Document parse(InputStream stream, String systemId)

Parses the contents of the given `java.io.InputStream` as an XML document and returns an `org.w3c.dom.Document` object. The systemId provides a base for resolving relative URIs.

Throws a `java.io.IOException` if any IO errors occur reading the given `InputStream`.

Throws an `IllegalArgumentException` if the given `InputStream` is null.

4.4.4 public Document parse(String uri)

Parses the content at the location specified by the given URI as an XML document and returns an `org.w3c.dom.Document` object.

Throws a `java.io.IOException` if any IO errors occur while reading the content specified by the URI.

Throws an `IllegalArgumentException` if the URI is null.

4.4.5 public Document parse(File file)

Parses the content of the given `java.io.File` as an XML document and returns an `org.w3c.dom.Document` object.

Throws a `java.io.IOException` if any IO errors occur while reading the content from the `File`.

Throws an `IllegalArgumentException` if the `File` is null.

4.4.6 public abstract Document parse(InputSource source)

Parses the content of the given `org.xml.sax.InputSource` as an XML document and returns a `org.w3c.dom.Document` object.

Throws a `java.io.IOException` if any IO errors occur reading the content from the `InputSource`.

Throws an `IllegalArgumentException` if the `InputSource` is null.

4.4.7 public abstract boolean isNamespaceAware()

Returns whether or not this parser supports XML namespaces.

4.4.8 public abstract boolean isValidating()

Returns whether or not this parser supports validating XML content as it is parsed.

4.4.9 public abstract void setEntityResolver(EntityResolver er)

Specifies the `org.xml.sax.EntityResolver` to be used by this `DocumentBuilder`. Setting the `EntityResolver` to null, or not calling this method, will cause the underlying implementation to use its own default implementation and behavior.

4.4.10 public abstract void setErrorHandler(ErrorHandler eh)

Specifies the `org.xml.sax.ErrorHandler` to be used by this `DocumentBuilder`. Setting the `ErrorHandler` to null, or not calling this method, will cause the underlying implementation to use its own default implementation and behavior.

4.4.11 public Document newDocument()

Creates an new `org.w3c.dom.Document` instance from the underlying DOM implementation.

4.5 public abstract class TransformFactory

The `TransformFactory` defines a factory API that enables applications to obtain a Transform object.

```
public abstract class TransformFactory {
    public static TransformFactory newInstace();
    protected TransformFactory();
    public abstract Transform newTransform(
                                  java.io.File stylesheet);
    public abstract Transform newTransform(
                               java.io.InputStream stylesheet);
    public abstract Transform newTransform(String url);
    public abstract void setAttribute(String name,
                                      Object value);
    public abstract Object getAttribute(String name);
}
```

4.5.1 public static TransformFactory newInstance()

Returns a new instance of a `TransformFactory`. Every call to `newInstance` will return a unique instance of `TransformFactory`.

Throws a `FactoryConfigurationError` if the class implementing the factory cannot be found or instantiated. An Error is thrown instead of an exception because the application is not expected to handle or recover from such events.

4.5.2 protected TransformFactory()

An empty constructor is provided. Implementors of this abstract class must provide their own public no-argument constructor in order for the static `newInstance` method to work correctly. Application programmers should be able to instantiate an implementation of this abstract class directly if they want to use a specific implementation of this API without using the static newInstance method to obtain the configured or platform default implementation.

4.5.3 public abstract Transform new Transform(java.io.File stylesheet)

Returns a new instance of type `Transform using the given stylesheet.`

Implementation of the `Transform` class is not affected by subsequent changes in their factories configuration.

4.5.4 public abstract Transform new Transform(java.io.InputStream stylesheet)

Returns a new instance of type `Transform using the given stylesheet.`

4.5.5 public abstract Transform new Transform(java.lang.String url)

Returns a new instance of type `Transform using the given stylesheet at the location pointed by the url.`

4.5.6 public abstract void setAttribute(String name, Object value)

Allows the user to set specific attributes provided by the underlying implementation. It is recommended that the name used to describe these attributes be unique and one way to achieve this could be to use the package naming convention. The names beginning with java and javax are reserved by the specification.

Throws `IllegalArgumentException` if the underlying implementation doesn't recognize the attribute.

4.5.7 public abstract Object getAttribute(String name)

Returns the attribute value.

Throws IllegalArgumentException if the underlying implementation doesn't recognize the attribute.

4.6 public abstract class Transform

Implements a Transform based on an XSLT stylesheet. An instance of this class can be obtained from the TransformFactory.newTransform method. Once an instance of this class is obtained, XML can be processed from a variety of sources with the output from the transform being written to a variety of sinks.

```
public abstract class Transform {
    protected Transform();
```

```
    public abstract void transform (java.io.File in,
                                    java.io.File out)
          throws SAXException, IOException;
    public abstract void transform (java.io.InputStream in,
                                    java.io.OutputStream out,
                                    String systemId);
          throws SAXException, IOException;
    public abstract void setEntityResolver(EntityResolver er);
    public abstract void setErrorHandler(ErrorHandler eh);
    public abstract void setXSLTParam(String name, Object
value);
    public abstract Object getXSLTParam(String name);
}
```

4.6.1 protected Transform()

An empty constructor is provided. Implementations should provide a protected constructor so that their factory implementation can instantiate instances of the implementation class. Application programmers should not be able to directly construct implementation subclasses of this abstract subclass. The only way a application should be able to obtain a reference to a `Transform` implementation instance is by using the appropriate methods of the `TransformFactory`.

4.6.2 public abstract void transform (java.io.File in, java.io.File out)

Applies the transform to the contents of the given input `File` and writes the result to the given output `File`.

Throws a `java.io.IOException` if any IO errors occur reading or writing to the given `Files`.

4.6.3 public abstract void transform (java.io.InputStream in, java.io.OutputStream out, String systemId)

Applies the transform to the contents of the InputStream and returns the result via the OutputStream. The systemId provides a base for resolving relative URIs.

Throws a `java.io.IOException` if any IO errors occur reading or writing to the given `Streams`.

4.6.4 public abstract void setEntityResolver(EntityResolver er)

Specifies the `org.xml.sax.EntityResolver` to be used by this `Transform`. Setting the `EntityResolver` to null, or not calling this method, will cause the underlying implementation to use its own default implementation and behavior.

4.6.5 public abstract void setErrorHandler(ErrorHandler eh)

Specifies the `org.xml.sax.ErrorHandler` to be used by this `Transform`. Setting the `ErrorHandler` to null, or not calling this method, will cause the underlying implementation to use its own default implementation and behavior.

4.6.6 public abstract void setXSLTParam(String name, Object value)

Allows the user to specify parameters for use by the stylesheet. This corresponds to the xsl:param in stylesheets.

4.6.7 public abstract Object getXSLTParam(String name)

Returns the value of the parameter specified by the name.

4.7 public class FactoryConfigurationError

This error is thrown if there is a configuration problem when creating new factory instances. This error will also be thrown when the class of a Factory specified by a system property, or the class of the default system parser factory, cannot be loaded or instantiated. Implementation or Application developers should never need to directly construct or catch errors of this type.

```
public class FactoryConfigurationError extends Error {
    public FactoryConfigurationError();
    public FactoryConfigurationError(String msg);
    public FactoryConfigurationError(Exception e);
    public FactoryConfigurationError(Exception e, String msg);
    public String getMessage();
    public Exception getException();
}
```

4.7.1 public FactoryConfigurationError()

Constructs a new `FactoryConfigurationError` with no detail message.

4.7.2 public FactoryConfigurationError(String msg)

Constructs a new `FactoryConfigurationError` with the given detail message.

4.7.3 public FactoryConfigurationError(Exception e)

Constructs a new `FactoryConfigurationError` with the given `Exception` as a root cause.

4.7.4 public FactoryConfigurationError(Exception e, String msg)

Constructs a new `FactoryConfigurationError` with the given `Exception` as a root cause and the given detail message.

4.7.5 public String getMessage()

Returns the detail message of the error or null if there is no detail message.

4.7.6 public Exception getException()

Returns the root cause of the error or null if there is none.

4.8 public class ParserConfigurationException

This exception is thrown if a factory cannot configure a parser given its current configuration parameters. For example, if a parser factory cannot create parsers that validate, but have been configured to do so, it will throw this exception when a parser is requested to via the parser creation methods. Application developers are not expected to construct instances of this exception type, but must catch them in code that obtains parser instances from a factory.

```
public class ParserConfigurationException extends Exception {
    public ParserConfigurationException();
    public ParserConfigurationException(String msg);
}
```

4.8.1 public ParserConfigurationException()

Constructs a new `ParserConfigurationException` with no detail error message.

4.8.2 public ParserConfigurationException(String msg)

Constructs a new `ParserConfigurationException` with the given detail error message.

SECTION 5

Conformance Requirements

This section describes the conformance requirements for parser implementations of this specification. Parser 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 this specification need to be conformant as per Section 5 of the XML 1.0 recommendation (http://www.w3.org/TR/1998/REC-xml-19980210), Section 6 of the XML Namespaces recommendation (http://www.w3.org/TR/REC-xml-names/) and Section 17 of the XSLT recommendation (http://www.w3.org/TR/xslt). Parsers that support validation only need to support DTDs. In addition to the above, implementations of the SAX 2.0, SAX2.0 extensions and DOM Level 2 core interfaces must be supported.

SECTION 6

Change History

This section lists the changes that have occurred over the development of this specification.

6.1 From 1.0 Final Release to 1.1 Public Review

Added parameter systemId to all the parse and transform methods which take Streams as parameters. This was done to provide a base to resolve relative URIs.

Added setIgnoreWhitespace, setExpandEntityReference, setIgnoringComments, setAttributes and the corresponding getters to DocumentBuilderFactory.

Added get/setAttribute to TransformFactory.

Added setEntityResolver, setErrorHandler and get/setXSLParam to Transform.

Added get/setFeature to SAXParserFactory.

Added get/setProperty to SAXParser.

Added SAX2 extensions.

Added Transformations

Added more mechanisms to look up an implementation of the various factories..

Removed conformance requirements from the specification and just refer to the conformance requirements as required by the specifications included by reference.

6.2 From 1.0 Public Release to 1.0 Final Release

The reservation of the java and javax namespace prefixes was removed. The XML Namespace specification is clear that a namespace is a collection of names that is identified by a URI reference. The prefix is a local identifier for the URI reference, therefore the reservation of the java and javax namespaces was in error.

6.3 From 1.0 Public Review to 1.0 Public Release

From the Public Review draft of this specification to the Public Release version, the specification was reordered and rewritten to address general feedback from the user community. This feedback indicated that the specification was too detailed in describing the endorsed specifications and not detailed enough in describing the plugability layer.

The `newParser` method of the `SAXParserFactory` abstract class was removed. Feedback showed that it was confusing to be able to obtain both the `SAXParser` wrapper and the underlying implementation from the factory. Removing this method allows the API to be more understandable while preserving the ability to access the underlying parser via the `getParser` method of the `SAXParser` abstract class.

The `getLocale` and `setLocale` methods of the various classes were removed. Instead it was felt that parser implementation authors should report errors in the configured default locale of the execution environment.

A new exception named `ParserConfigurationException` was added so that a parser factory can signal to an application that it can't provide a parser with the desired configuration. The `checkXXX` methods aren't sufficient for this purpose as a situation may arise where there is a mutually exclusive setting of various parser

properties. At this time, this problem is potentially minor as there are only two settable properties on each of the parser types, but in the future as the number of settable properties increases, the problem would get harder to solve without an exception that could be thrown at parser creation time. As part of this change, the `setXXX` property methods of the factories no longer throw an `IllegalArgumentException` if they are set to a property which cannot be supported.

The `FactoryException` class was renamed to `FactoryConfigurationError`. This rename was undertaken to emphasize that such an error condition is a fatal condition that an application should not be reasonable expected to handle.

SECTION 7

Future Directions

This version of the Java API for XML Processing includes the basic facilities for working with XML documents using either the SAX, DOM and XSLT APIs. However, there is always more to be done.

This section briefly describes our plans for future versions of this specification. Please keep in mind that the items listed here are preliminary and there is no commitment to the inclusion of any specific feature in any specific version of the specification. In addition, this list of items is by no means the only features that may appear in a future revision. Your feedback is encouraged.

7.1 Updated SAX and DOM Support

As future versions of SAX and DOM evolve it will be incorporated into the future version of this API.

7.2 Update XSL Plugability Support

XSL (eXtensible Stylesheet Language) is a language for expressing stylesheets that can be used with XML document. It consists of two parts:

- A language for transforming XSL documents (also known as XSLT)
- An XML vocabulary for specifying formatting specfics

XSL Transformations has been formalized as a W3C Recommendation. In a future version of the specification, we would like to provide a plugability API to allow an application programmer to provide an XML document and an XSLT document to a wrapped XSLT processor and obtain a transformed result.

7.3 Plugability Mechanism Enhancements

Various ways of making the plugability mechanism work have been incorporated into this version of the spec. However if there are other ways in the future to enhance this, it will be included in the future versions of this API.