Java API for XML Parsing

Version 1.0 Final Release

James Duncan Davidson (et al)

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

March 2, 2000

JavaTM API for XML Parsing Specification ("Specifica-tion")
Version: 1.0
Status: FCS
Release: March 2, 2000

901 San Antonio Road, Palo Alto, California 94303, U.S.A.

NOTICE

TRADEMARKS

DISCLAIMER OF WARRANTIES

LIMITATION OF LIABILITY

RESTRICTED RIGHTS LEGEND

REPORT

Contents

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 Febuary 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 simplier 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 technolgies 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 Parsing, Version 1.0. 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 Compatibility Test Suite will also be available that will verify whether an implementation of this specification is compliant.

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, or SAX. 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. It was developed under the authorization of Java Specification Request 5. More information about the Java Community Process can be found at:

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

The specific information contained in Java Specification Request 5 can be found at:

`http://java.sun.com/aboutJava/communityprocess/jsr/jsr_005_xml.html`

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

- James Duncan Davidson (Spec Lead), Sun
- Larry Cable (Original Spec Lead), Sun
- Takuki Kamiya, Fujitsu Ltd
- Scott Dietzen, BEA Weblogic
- Jon Winston, Ariba
- David Booth, Bluestone
- Mark Scardina, Oracle
- A. L. N. Reddy, Netscape
- Scott Fairchild, MCI WorldCom
- Kevin Lawrence, IBM

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:

`xml-spec-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-5 Expert Group listed above
- Graham Hamilton, Eduardo Pelegri-Lopart, Rajiv Mordani, Mark Hapner, Connie Weiss, Nancy K. Lee, Mark Reinhold, Josh Bloch, and Bill Shannon all of whom work at Sun Microsystems and whose talents have all reflected upon the development of this API.
- David Brownell who led Sun's early investigations into Java Platform based XML explorations.
- Larry Cable who started the JSR-5 process and led the development of the API though early Public Draft Phase.
- Eric Armstrong, Pier Fumagalli and Jason Hunter for reviewing late copies of this spec and finding many areas which needed correction or clarification.

Most importantly, everyone who sent in feedback to this document and who commented on Sun's Project X technologies which served as a starting point for developing this specification.

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 Compatibility Test Suite 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 subsumes 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 subsumes the XML Namespaces 1.0 Recommendation in its entirety.

2.3 Simple API for XML Parsing (SAX)

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 1.0 API is located at:

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

This specification subsumes the SAX 1.0 API in its entirety. The API packages subsumed are:

- `org.xml.sax.*`
- `org.xml.sax.helpers.*`

2.4 Document Object Model (DOM) Level 1

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 1 specification defines these interfaces using Interface Definition Language (IDL) in a language independent fashion and also includes a Java Language binding.

The DOM Level 1 Recommendation is located at:

`http://www.w3.org/TR/REC-DOM-Level-1/`

This specification subsumes both the abstract semantics described for the DOM Level 1 Core interfaces and the associated Java Language binding. It does not subsume the HTML-based extensions defined by the Recommendation. The API package subsumed by this specification is:

- `org.w3c.dom.*`

SECTION 3

Plugability Layer

The SAX and DOM 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 SAX and DOM functionality 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.HandlerBase` API to a `SAXParser` implementation and parse XML documents. As the parser processes the XML document, it will call methods on the provided `HandlerBase`.

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 `javax.xml.parsers.SAXParserFactory` system property to determine the `SAXParserFactory` implementation class to load, instantiate and return. If the `javax.xml.parsers.SAXParserFactory` system property is not defined, then a platform default `SAXParserFactory` instance will be returned.

If the `SAXParserFactory` implementation class described by the `javax.xml.parsers.SAXParserFactory` property cannot be loaded or instantiated at runtime, a `FactoryConfigurationError` is thrown. This error message must 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;
HandlerBase handler = new MyApplicationHandlerBase();
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;
HandlerBase handler = new MyApplicationHandlerBase();
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
}
```

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 `javax.xml.parsers.DocumentBuilderFactory` system property to determine the `DocumentBuilderFactory` implementation class to load, instantiate and return. If the `javax.xml.parsers.DocumentBuilderFactory` system property is not defined, then a platform default `DocumentBuilderFactory` instance will be returned.

If the `DocumentBuilderFactory` implementation class described by the `javax.xml.parsers.DocumentBuilderFactory` property cannot be loaded or instantiated at runtime, a `FactoryConfigurationError` is thrown. This error message must 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
}
```

3.3 Thread Safety

Implementations of the `SAXParser` and `DocumentBuilder` 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` or `DocumentBuilder` 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 on thread has access to any given `SAXParser` or `DocumentBuilder` instance.

Configuration of a `SAXParserFactory` or `DocumentBuilderFactory` 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 and the `newDocumentBuilder` method of a `DocumentBuilderFactory` 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

Package javax.xml.parsers

This section defines the API of the `javax.xml.parsers` package.

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 {
    public static SAXParserFactory newInstace();
    protected SAXParserFactory();
    public SAXParser newSAXParser()
        throws ParserConfigurationException, SAXException;
    public void setNamespaceAware(boolean aware);
    public void setValidating(boolean validating);
    public boolean isNamespaceAware();
    public boolean isValidating();
}
```

4.1.1 public static SAXParserFactory newInstance()

Returns a new instance of a `SAXParserFactory`. The implementation of the `SAXParserFactory` returned depends on the setting of the `javax.xml.parsers.SAXParserFactory` property or, if the property is not set, a platform specific default.

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.

4.1.2 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.

4.1.3 public SAXParser newSAXParser()

Returns a new configured instance of type `SAXParser`.

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

4.2 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 void parse(InputStream stream, HandlerBase base)
        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 abstract org.xml.sax.Parser getParser()
        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 void parse(InputStream stream, HandlerBase base)

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

4.2.3 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.

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.

4.2.4 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.

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.

4.2.5 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.

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.

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

Returns the underlying `org.xml.sax.Parser` object which is wrapped by this `SAXParser` implementation.

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

4.2.7 public abstract boolean isNamespaceAware()

Returns whether or not this parser supports XML namespaces.

4.2.8 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 {
```

```
    public static DocumentBuilderFactory newInstance();
    protected DocumentBuilderFactory();
    public DocumentBuilder newDocumentBuilder()
        throws ParserConfigurationException;
    public void setNamespaceAware(boolean awareness);
    public void setValidating(boolean validating);
    public boolean isNamespaceAware();
    public boolean isValidating();
}
```

4.3.1 public static DocumentBuilderFactory newInstance()

Returns a new instance of a `DocumentBuilderFactory`. The implementation of the `DocumentBuilderFactory` returned depends on the setting of the `javax.xml.parsers.DocumentBuilderFactory` property or, if the property is not set, a platform specific default.

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.

4.3.2 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.

4.3.3 public DocumentBuilder newDocumentBuilder()

Returns a new configured instance of type `DocumentBuilder`.

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 boolean isNamespaceAware()

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

4.3.7 public boolean isValidating()

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

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(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(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.

4.4.4 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.

4.4.5 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.

4.4.6 public abstract boolean isNamespaceAware()

Returns whether or not this parser supports XML namespaces.

4.4.7 public abstract boolean isValidating()

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

4.4.8 public abstract void setEntityResolver(EntityResolver er)

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

4.4.9 public abstract void setErrorHandler(ErrorHandler eh)

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

4.4.10 public Document newDocument()

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

4.5 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();
}
```

4.5.1 public FactoryConfigurationError()

Constructs a new `FactoryConfigurationError` with no detail message.

4.5.2 public FactoryConfigurationError(String msg)

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

4.5.3 public FactoryConfigurationError(Exception e)

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

4.5.4 public FactoryConfigurationError(Exception e, String msg)

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

4.5.5 public String getMessage()

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

4.5.6 public Exception getException()

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

4.6 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);
}
```

4.6.1 public ParserConfigurationException()

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

4.6.2 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.

5.1 Document Character Set Encoding Conformance

XML documents, both markup and content, are represented using the UNICODE character set. A character set may be physically encoded using one or more character set encodings. An XML document's encoding is typically announced in the prolog of the document in the XML declaration Processing Instruction. For example:

```
<?XML Version="1.0" encoding="UTF-8"?>
```

Note that if the XML document's character encoding is ASCII, the XML declaration does not need to contain the encoding attribute. Appendix F of the W3C XML 1.0 Recommendation describes a mechanism for determining the character encoding of an XML document that is not encoded in the UTF-8 or UTF-16 character sets.

Parser implementations are required to support the following character set encodings:

- ASCII
- UTF-8
- UTF-16

In addition, parser implementations may optionally support additional encodings, including the following encoding values which are also defined in the W3C XML Recommendation:

- ISO-10646-UCS-2
- ISO-10646-UCS-4
- ISO-8859-1
- ISO-8859-2
- ISO-8859-3
- ISO-8859-4
- ISO-8859-5
- ISO-8859-6
- ISO-8859-7
- ISO-8859-8
- ISO-8859-9
- ISO-2022-JP
- Shift_JIS
- EUC-JP

It is an error for a document to declare a particular encoding and actually use another. If this situation occurs, the parser must throw an exception of type `SAXException` and stop processing the document.

Parser implementations are required to support the facility whereby an external entity may declare its own encoding distinct from that of the referencing entity or document.

5.2 Well Formedness Conformance

The W3C XML 1.0 Recommendation defines a well formed document to be a textual object that meets the following requirements:

- There is exactly one element, the root (or document) element which may contain other elements.
- Meets all the well-formedness constraints defined in the Recommendation
- References, either directly or indirectly, only parsed entities that are also well formed.

All parser implementations implementing this specification are required to report any violations of the well-formedness constraints defined by the W3C XML 1.0 Recommendation.

5.3 Validity Conformance

In addition to checking documents for well-formedness, as defined above, a validating parser implementation is also required to check an XML document for conformance to:

- The Document's DTD (if any)
- The XML validity constraints defined in section 2.8 of the W3C XML 1.0 Recommendation

5.4 XML Namespace Conformance

Parser implementations may optionally provide support to parse documents that utilize the W3C XML Namespaces Recommendation.

5.4.1 Non Validating Parser Conformance

A non-validating parser that implements namespace support is required to check for, and report as an error, any syntactic violations defined by the W3C XML Namespaces Recommendation. Parser implementations are required to detect namespace usage that has no matching prior namespace declaration, either within the body of the document entity or within the internal subset of a document's DTD.

Parser implementations encountering namespace usage without a prior matching namespace declaration shall result in a parsing error.

5.4.2 Validating Parser Conformance

In addition to meeting the requirements for a non-validating parser detailed above, a validating parser that implements namespace support as defined is required to check for, and report as an error, any namespace used buy not declared within a document or its internal or external DTD sets.

SECTION 6

Change History

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

6.1 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.

6.2 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 first version of the Java API for XML Parsing includes the basic facilities for working with XML documents using either the SAX or DOM 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 this specification was finalized, the SAX2 and DOM Level 2 specifications were well on their way to completion. It is anticipated that these revisions to the SAX and DOM specifications will be completed in time to be reflected in the next version of this API.

7.2 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 Pluggability Mechanism Enhancements

While system properties are a useful mechanism for allowing pluggability of parsers, they do not cover some common use cases. Future versions of this specification need to provide alternative pluggability mechanisms for these cases.