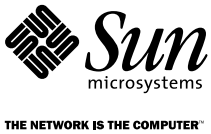
Java API for XML Parsing Specification

Version 1.0 [Public Draft 1]

please send comments to xml-spec-comments@eng.sun.com



THE NETWORK IS THE COMPUTER™

Java Software
A Division of Sun Microsystems, Inc.
901 San AntonioRoad
Palo Alto, California 94303
415 960-1300 fax: 415 969 9131

November 16, 1999

Larry Cable

Java™ API for XML Parsing Specification ("Specification")
Version: 1.0
Status: Public Draft 1
Release: November 22nd, 1999

NOTICE
The Specification is protected by copyright and the information described therein may be protected by one or more U.S. patents, foreign patents, or pending applications. Except as provided under the following license, no part of the Specification may be reproduced in any form by any means without the prior written authorization of Sun Microsystems, Inc. ("Sun") and its licensors, if any. Any use of the Specification and the information described therein will be governed by the terms and conditions of this license and the Export Control and General Terms as set forth in Sun's website Legal Terms. By viewing, downloading or otherwise copying the Specification, you agree that you have read, understood, and will comply with all of the terms and conditions set forth herein.

Subject to the terms and conditions of this license, Sun hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense) under Sun's intellectual property rights to review the Specification internally for the purposes of evaluation only. Other than this limited license, you acquire no right, title or interest in or to the Specification or any other Sun intellectual property. The Specification contains the proprietary and confidential information of Sun and may only be used in accordance with the license terms set forth herein. This license will expire ninety (90) days from the date of Release listed above and will terminate immediately without notice from Sun if you fail to comply with any provision of this license. Upon termination, you must cease use of or destroy the Specification.

TRADEMARKS
No right, title, or interest in or to any trademarks, service marks, or trade names of Sun or Sun's licensors is granted hereunder. Sun, Sun Microsystems, the Sun logo, Java, the Coffee Cup logo and Duke logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

DISCLAIMER OF WARRANTIES
THE SPECIFICATION IS PROVIDED "AS IS" AND IS EXPERIMENTAL AND MAY CONTAIN DEFECTS OR DEFICIENCIES WHICH CANNOT OR WILL NOT BE CORRECTED BY SUN. SUN MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE OR THAT ANY PRACTICE OR IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS 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
Use, duplication, or disclosure by the U.S. Government is subject to the restrictions set forth in this license and as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii)(Oct 1988), FAR 12.212(a) (1995), FAR 52.227-19 (June 1987), or FAR 52.227-14(ALT III) (June 1987), as applicable.

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.

Contents

Preface

This is the Java API for XML Parsing *1.0 Specification.* This document describes the APIs available in the version 1.0 of this specification.

Details on the conditions under which this document is distributed are described in the license herein.

Due to the volume of interest in XML, we cannot normally respond individually to reviewer comments, but we carefully read and consider all reviewer input. Please send comments to xml-spec-comments@eng.sun.com

To stay in touch with the XML project, visit our web site at:

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

Who should read this document

This document is intended for:

- Application Developers wishing to develop portable Java[tm] Language applications that use XML APIs.
- Java[tm] Platform Developers wishing to implement this version of the Standard Extension.

This document is not a User's Guide.

Related Documents

This Specification depends upon, and references or subsumes, all or part of several

specifications produced by the World Wide Web Consortium and other standards bodies.

TABLE P-1 Normative References

XML Specification	http://www.w3.org/TR/1998/REC-xml-19980210
DOM Level 1 Specification	http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/
DOM Level 1 errata	http://www.w3.org/DOM/updates/REC-DOM-Level-1-19981001-errata.html
XML Namespaces	http://www.w3.org/TR/1999/REC-xml-names-19990114
ISO 10646	ISO (International Organization for Standardization). ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).
Unicode	The Unicode Consortium. The Unicode Standard, Version 2.0. Reading, Mass.: Addison-Wesley Developers Press, 1996.

TABLE P-2 Non-normative References

HTML 4.0	http://www.w3.org/TR/1998/REC-html40-19980424/
CORBA 2.2	http://www.omg.org/corba/corbaiiop.html

Related Copyrights

This specification either directly includes or references copyrighted materials from other sources. In compliance with the terms of those copyright(s), they are reproduced here in their entirety.

SAX

SAX has no copyright associated with it; it is in the public domain. Inclusion of this material into this specification is neither intended to, nor does affect in any way the status of SAX.

For the purposes of this specification, the definition of compliance, and subsequent claims of compliance, compliant implementations are required to implement (at least) the definition of SAX described herein.

W3C Copyright

DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright

in this document will at all times remain with copyright holders.

CORBA

This specification makes no direct reference to or inclusion of CORBA; the W3C DOM Level 1 specification makes use of CORBA to describe the DOM interfaces in a language independent fashion. No OMG copyrighted material is directly contained or referenced within this document.

Unicode

This specification makes no direct reference to or inclusion of Unicode; the W3C specification(s) make use of Unicode. No Unicode copyrighted material is directly contained or referenced within this document.

ISO 10646

This specification makes no direct reference to, or inclusion of ISO10646; the W3C specification(s) make use of ISO10646. No ISO copyrighted material is directly contained or referenced within this document.

Future Directions

Future revisions of this specification will include DOM Level 2 and SAX 2.0 support.

Acknowledgments

The success of the Java Platform depends on the process used to define and refine it. This open process permits the development of high quality specifications in internet time and involves many individuals and corporations.

Many people contributed to this specification, its reference implementation, and the specification(s) and implementation(s) it references.

Thanks to:

- The W3C DOM Working and Interest Group(s) for their work on the DOM Level 1 specification.
- David Megginson and the many contributors to the XML-DEV mailing lists that defined the SAX 1.0 implementation.
- The following people from Sun Microsystems: Eduardo Pelegri-Lleopart, Mala Chandra, Graham Hamilton, Mark Hapner, Rajiv Mordani, Connie Weiss, Nancy Lee, Mark Reinhold, Josh Bloch, and Bill Shannon.
- The members of the JCP expert group (in particular Takuki Kamiya of Fujitsu Ltd and Kelvin Lawrence of IBM) and participants who reviewed this document.

A special thank you to David Brownell (ex Sun Microsystems Inc.) who both championed XML here at Sun and authored much of Project X technology upon which the Reference Implementation is based, including the parser -- arguably the fastest, most compliant one around at the time of this writing.

Last, but certainly not least important, thanks to the software developers and members of the general public who have read this specification, used the reference implementation, and shared their experiences.

CHAPTER 1

Overview

1.1 XML

The *eXtensible Markup Language* is a meta-language defined by the World Wide Web Consortium (W3C), derived by them from the ISO Standard General Markup Language, that can be used to describe a broad range of hierarchical markup languages.

The XML “language” is described and defined in the W3C Standard for XML 1.0 document:

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

This specification subsumes that standard in its entirety for the purposes of defining the XML language manipulated by the APIs defined herein.

A “Markup Language” is a language that can be used to “mark up”, or annotate, arbitrary character data to describe the structure of and/or attribute meta-data to that character data. An XML “document” consists of a prolog, an optional DOCTYPE declaration, processing instruction(s), a root element (with optional attributes) and a hierarchy of sub-elements (optionally with attributes), entities, and (parsed) character data.

XML uses the ISO 10646 character set and may be encoded using a variety of encodings, including UTF-8 and UTF-16. This specification does not modify the W3C standard with regard to character set or encoding.

An XML document may be “well-formed” and may also be “valid”.

- A “well formed” XML document conforms to the “well-formed-ness” constraints defined by the XML specification.
- A “valid” XML document is “well formed” and is validated against a Document Type Definition (DTD) as defined by the “validity” constraints defined by the XML specification.

This specification does not affect the definitions of either “well-formedness” or “validity” as defined in the W3C specification.

XML is being used for a broad variety of applications including:

- “vertical” markup languages for mathematics, chemistry, and other document-centric publishing applications
- E-Commerce solutions
- (intra, extra, and inter) enterprise application messaging.

This version of the Standard Extension is intended to introduce basic support for parsing and manipulating XML documents through Java APIs.

1.2 XML Parser

An XML Parser is a software engine[1] that is capable of the following:

- Consuming a stream of characters, suitably encoded.
- Tokenizing that character stream into XML syntactic constructs.
- “Parsing” that tokenized stream to determine if it is “well-formed” and “valid.”
- Potentially exposing the parsed document’s structure, and/or its “well-formed-ness” or “validity” to some “client” of the parser.

This specification does not mandate a particular parser implementation API. It only mandates the existence of a parser, its conformance requirements, and a simple API for “application” code. The API enables applications to access a parser implementation, to invoke that parser on an XML document represented as a character stream, and to expose the structure and/or “well-formed-ness” and “validity” in an implementation-independent fashion.

1.3 DOM

The Document Object Model (or DOM) is a set of interfaces defined by the DOM Working Group of the World Wide Web Consortium describing facilities for a programmatic representation of a parsed XML (or HTML) document. The DOM Level 1 specification defines these interfaces using CORBA IDL in a language-independent fashion, but also includes a Java(tm) Language binding. This specification subsumes both the abstract semantics described for the DOM Level 1 (Core) interfaces and the associated Java(tm) Language binding.

This specification does not subsume the HTML extensions defined by the DOM Level 1 specification.

1. For the purposes of this specification, a conforming parser shall either be implemented purely in Java or callable from Java via the JNI facility.

1.4 SAX

The Simple API for XML (or SAX) is an API in the public domain, developed by many individuals on the XML-DEV mailing list, that provides an event-driven interface to the process of parsing an XML document.

An event driven interface provides a mechanism for “callback” notification(s) to “application” code as the underlying parser recognizes XML syntactic constructions in the document it is parsing on behalf of the application.

SAX does not have a formal specification document; it is defined by a public domain API implementation using the Java™ Programming Language.

This specification formally defines, and thus subsumes, that API as defined by the implementation for SAX Version 1.0.

1.5 XML Namespaces

The XML Namespaces Specification 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 (elements, attributes, entities) may have a distinguishing “namespace” associated with it, and the responsibility of XML parsers in handling and exposing such namespace information.

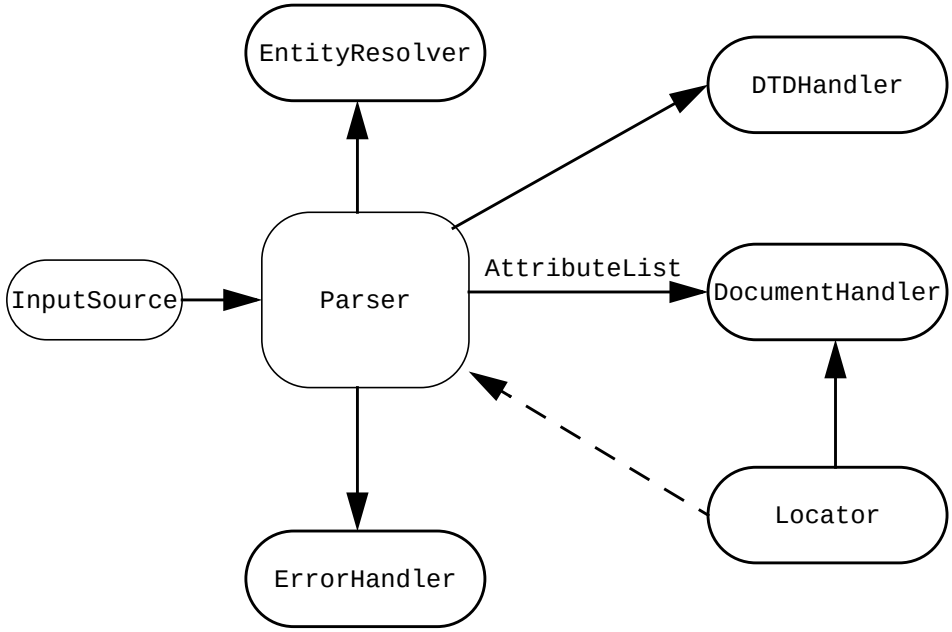
This specification subsumes the content of the W3C XML Namespace specification.

CHAPTER **2**

SAX

2.1 Overview

The Simple API to XML (or SAX) is an event-driven XML Parsing API. This version of the specification describes SAX version 1.0.



The ‘client’ of a SAX `Parser` provides the `Parser` with an `InputSource`, encapsulating the XML to parse, and minimally also a `DocumentHandler`. The `Parser` consumes the content of the `InputSource` and delivers parsing ‘events’ to the client’s `DocumentHandler` as it encounters XML constructs in the content.

The `Parser` represents any attribute(s) and their associated value(s) it parses in an element occurrence to the client as an `AttributeList`.

If the client wishes to handle errors encountered by the `Parser`, it provides the `Parser` with an `ErrorHandler`. The `Parser` will notify the `ErrorHandler` of any errors that occur. In the absence of a client-supplied `ErrorHandler`, the `Parser` uses an implementation-dependent handler.

If the client wishes to receive notifications of any notations or unparsed entities the `Parser` may encounter during parsing, the client should register a `DTDHandler` with the `Parser`.

A `Parser` registers a `Locator` with the `DocumentHandler` to enable the handler to determine the current end position of the `Parser` in the XML content during `DocumentHandler` callbacks.

2.2 API Definition(s)

2.2.1 org.xml.sax.AttributeList

Description

The `org.xml.sax.AttributeList` interface represents a list of an XML element’s actual attribute(s) and associated value(s).

When a `Parser` encounters and successfully parses an element with associated attribute(s), it invokes its current `DocumentHandler.startElement()` method. It passes a reference to an `AttributeList` encapsulating the current element’s attribute(s) and associated value(s).

The `AttributeList` and its content are only valid for the duration of the `DocumentHandler.startElement()` invocation. Clients may not retain references to the `AttributeList` or its content thereafter.

For a copy of an `AttributeList` to persist beyond the invocation of `DocumentHandler.startElement()`, an application may do one of the following:

- Test the `AttributeList` to determine if it implements `java.lang.Cloneable` and invoke `Object.clone()` to create a copy.
- Use some other data structure (such as `java.util.Hashtable`) and copy the attribute value association(s) explicitly.

An `AttributeList` instance only enumerates attribute(s) and associated value(s) that were actually specified or defaulted for the current element. `#IMPLIED` attributes not actually occurring in the current element are never enumerated in an `AttributeList`. (Their absence from the `AttributeList` should imply their value.)

A `Parser` implementation may provide the contents of an `AttributeList` in *any* arbitrary order. It is not require to list them in order of declaration (if any) or specification.

Methods

`int getLength()`	return the number of attributes in the `AttributeList` or 0
`String getName (int i)`	return the attribute's name by position in the `AttributeList` at index `i`, or `null` if index `i` where `(0 <= i < getLength())` is out of bounds. If the attribute name has a namespace prefix in the document parsed, then the name returned shall include that prefix to the local part.
`String getType (int i)`	return the attribute's type by position in the `AttributeList` at index `i`, or `null` if index `i` where `(0 <= i < getLength())` is out of bounds. The type returned is one of: • "CDATA" • "ID" • "IDREF" • "IDREFS" • "NMTONKEN" • "ENTITY" • "ENTITIES" • "NOTATION" If the `Parser` has not read a declaration for the attribute, or if the `Parser` does not report attribute types, then it must return the value "CDATA" as required by the XML 1.0 specification (clause 3.3.3, "Attribute-Value Normalization"). For an enumerated attribute that is not a notation, the type shall be "`NMTOKEN`".

`String getValue(int i)`	return the value by position in the `AttributeList` at index `i`, or `null` if index `i` where `(0 <= i < getLength())` is out of bounds. If the value of the attribute is a list of tokens (IDREFS, ENTITIES, or NMTOKENS) the tokens will be concatenated into a single `String` result, each token separated by at least one whitespace character.
`String getType(String name)`	return the type of the attribute with the specified `name` or `null` if no such attribute is present in the `AttributeList`. The return values are the same as those for the method `getType(int)` defined above.
`String getValue(String name)`	return the value of the attribute with the specified `name`, or `null`, if no such named attribute is present in the `AttributeList`. If the value of the attribute is a list of tokens (IDREFS, ENTITIES, or NMTOKENS) the tokens will be concatenated into a single `String` result, each token separated by at least one whitespace character.

2.2.2 org.xml.sax.DTDHandler

Description

The `org.xml.sax.DTDHandler` interface is typically implemented by client code. The client registers the interface with a `Parser` object via its `setDTDHandler()` method to receive notifications (callbacks) from that `Parser` when it encounters either notations or unparsed entities while parsing XML content.

A `Parser` may report unparsed entity and notation events to the `DTDHandler` in any order, regardless of the order in which they occur in the source document.

All DTD notifications shall be delivered to the `DTDHandler` after the `DocumentHandler.startDocument()` method, and before any `DocumentHandler.startElement()` methods.

Methods

void notationDecl(String name, String publicId, String systemId) throws SAXException	This is a callback to notify the client that the parser has encountered a notation declaration in the source document. The `name` parameter is the name of the declared notation. The `publicId` parameter is the public identifier associated with the notation, or `null` if none specified. The `systemId` parameter is the system identifier associated with the notation, or `null` if none specified.
void unparsedEntityDecl(String name, String publicId, String systemId, String notationName } throws SAXException	This is a callback to notify the client that the parser has encountered an unparsed entity declaration in the source document. The `name` parameter is the name of the unparsed entity. The `publicId` parameter is the public identifier associated with the unparsed entity, or `null` if none is specified. The `systemId` parameter is the system identifier associated with the unparsed entity. The notationName parameter is the name of the associated notation.

2.2.3 org.xml.sax.DocumentHandler

Description

The `org.xml.sax.DocumentHandler` interface is typically implemented by client code. The client registers it with a `Parser` using its `setDocumentHandler()` method to receive notifications (callbacks) from the `Parser` when it encounters XML markup, such as the (root) elements, processing instructions, whitespace, and content.

This is the primary client interface that a client uses, along with a `Parser` instance, to parse an XML source.

Note: Any given instance of a `DocumentHandler` may only be used with one instance of a `Parser` at any time to parse XML content. Using a single `DocumentHandler` with multiple `Parser` instances parsing simultaneously will result in errors.

Methods

`void startDocument() throws SAXException`	The `Parser` invokes this method exactly once, before any other in the `DocumentHandler` (except `setDocumentLocator()`) and `DTDHandler`. This method notifies the client that the parsing of an XML document has commenced. The client may signal a parsing error by throwing an appropriate `SAXException`. Exception behavior is implementation dependent; a particular `Parser` implementation may choose to either stop or continue parsing.
`void endDocument() throws SAXException`	The `Parser` invokes this method exactly once, as the last method invoked for a particular invocation of its `parse()` methods. This method notifies the client that the `Parser` has encountered the document end. The `Parser` shall not invoke this method until it has either abandoned parsing or reached the end of input. The client may signal a parsing error by throwing an appropriate `SAXException`. Exception behavior is implementation dependent; a particular `Parser` implementation may choose to either stop or continue parsing.

void ignorableWhitespace(char[] ch, int start, int length)throws SAXException	A validating `Parser` implementation uses this method to notify the client that it has encountered a contiguous block of ignorable whitespace characters in the XML content being parsed. Non-validating parsers may use this method if they are capable of parsing and using content models. A conforming `Parser` implementation may return all contiguous whitespace character data in either a single block/notification, or as multiple blocks/notifications. All of the whitespace characters in any single notification shall come from the same external entity, so that any `Locator` may provide valid information regarding the location of the whitespace character data. The `ch` parameter is an array of characters containing the character text (fragment) parsed. The `start` parameter is the index into the `ch` parameter array at which the whitespace text fragment parsed by the `Parser` begins. The `length` parameter is the number of whitespace characters in the `ch` parameter array, parsed by the `Parser`, that are the subject of this notification. It is illegal for the client to access whitespace characters in the `ch` parameter array beyond the bounds specified (`start`, `length`). Doing so results in unpredictable and non-portable behavior. The client may signal a parsing error by throwing an appropriate `SAXException`. Exception behavior is implementation dependent; a particular `Parser` implementation may choose to either stop or continue parsing.

`void startElement( String name, AttributeList atts ) throws SAXException`	The `Parser` invokes this method exactly once as it encounters each new element start tag in the XML input source (once per element). This method notifies the client that the `Parser` has encountered a new element usage. The `name` parameter is the actual element name. If the element name has an explicit namespace prefix associated with it, the `name` parameter shall contain that prefix as part of the name. The `atts` parameter is the `AttributeList` declared by the element (if any). If no attributes are declared, then `AttributeList.getLength()` returns `0`. The client may signal a parsing error by throwing an appropriate `SAXException`. Exception behavior is implementation dependent; a particular `Parser` implementation may choose to either stop or continue parsing.
`void endElement( String name ) throws SAXException`	The `Parser` invokes this method exactly once as it encounters each closing element end tag in the XML input source (once per element) to notify the client that it has encountered the end of an element usage. The name parameter is the actual element name of the closing (“end”) tag. If the element name has an explicit namespace prefix associated with it, the `name` parameter shall contain that prefix as part of the name. The client may signal a parsing error by throwing an appropriate `SAXException`. Exception behavior is implementation dependent; a particular `Parser` implementation may choose to either stop or continue parsing.

void characters(char[] ch, int start, int length)throws SAXException	The `Parser` invokes this method to notify the client that it has encountered a sequence of character text in the XML content. A conforming `Parser` implementation may return all contiguous character data in either a single block/ notification, or as multiple blocks/notifications. All of the characters in any single notification shall come from the same external entity, so that any `Locator` may provide valid information regarding the location of the character data. Non-validating parsers can report whitespace using either this method or the `ignorableWhitespace()` method. A validating Parser is required to report whitespace using the `ignorableWhitespace()` method. The `ch` parameter is an array of characters containing the character text (fragment) parsed. The `start` parameter is the index into the `ch` parameter array at which the character text fragment parsed by the `Parser` begins. The `length` parameter is the number of characters in the `ch` parameter array, parsed by the `Parser`, that are the subject of this notification. It is illegal for the client to access characters in the `ch` parameter array beyond the bounds specified (`start`, `length`). Doing so results in unpredictable and non-portable behavior. The client may signal a parsing error by throwing an appropriate `SAXException`. Exception behavior is implementation dependent; a particular `Parser` implementation may choose to either stop or continue parsing.

void processingInstruction(String target, String data)throws SAXException	A Parser invokes this method once for each processing instruction it encounters while parsing XML content. Note that processing instructions may appear before or after the main (root) document element. A conforming `Parser` implementation shall never report an initial XML declaration or a text declaration using this method. The `target` parameter is the processing instruction target. The `data` parameter is the data supplied with the Processing Instruction, or null if none was supplied. The client may signal a parsing error by throwing an appropriate `SAXException`. Exception behavior is implementation dependent; a `Parser` implementation may choose to either stop or continue parsing.
void setDocumentLocator(Locator l)	This method provides the implementing client with a `Locator` for the `Parser`. The `Parser` invokes this method before invoking any other `DocumentHandler` methods. The client can use the `Locator` from any of the `DocumentHandler` callback methods above to determine the end position of any document related event. Use of the `Locator` outside the scope of the `DocumentHandler` callback methods above is illegal. A conforming `Parser` is required to provide a `Locator`.

2.2.4 org.xml.sax.EntityResolver

Description

The `org.xml.sax.EntityResolver` interface is typically implemented by client code to provide customizable handling of entity resolution during XML parsing. Entity resolution includes resolution of external DTD subset(s) and all external entities except the top-level document entity itself.

An `EntityResolver` instance may be registered with a particular `Parser` instance using its `setEntityResolver()` method.

When a conforming `Parser` encounters an unresolved entity reference in the XML content, it will invoke its `EntityResolver` to obtain an `InputSource` containing the resolved entity.

Methods

`InputSource resolveEntity( String publicId, String systemId ) throws IOException, SAXExcption`	The `Parser` invokes this method to resolve an external entity it has encountered in the XML content it is currently parsing. The `publicId` parameter is the public identifier of the external entity to be resolved, or `null` if none was supplied. The `systemId` parameter is the system identifier of the external entity resolved. If this is a URI, conforming parsers must fully resolve this URI before invoking this method. This method returns an `InputSource` object referring to the resolution of the entity, or null to indicate that the `Parser` will open a regular URI connection to the `systemId`. The client may signal a parsing error by throwing an appropriate `SAXException`. Exception behavior is implementation dependent; a particular `Parser` implementation may choose to either stop or continue parsing. An `IOException` that occurs while trying to resolve an external entity shall be treated as a fatal error to a validating `Parser` implementation. Non-validating parsers may either ignore this exception or signal it to the invoker of the parser through the `ErrorHandler.error()` method.

2.2.5 org.xml.sax.ErrorHandler

Description

The `org.xml.sax.ErrorHandler` interface is typically implemented by client code to provide customizable error handling support during XML content parsing.

An `ErrorHandler` instance may be registered with a `Parser` using its `setErrorHandler()` method.

If an `ErrorHandler` is registered with a `Parser`, then the `Parser` reports all error(s) during parsing by invoking this interface instead of throwing the offending `Exception` from the `Parser` method that initiated the error-generating operation.

A conforming `Parser` implementation is not required to continue to provide useful information (to continue to parse the content and deliver notifications via the `DocumentHandler` until EOF) after an invocation of `fatalError()`.

Methods

`void warning( SAXParseException e )throws SAXException`	The `Parser` invokes this method to notify the client that a recoverable condition has occurred during parsing. The default behavior of this method is to take no action. (In some circumstances, the warning may be conveyed to a human user via some error messaging facility.) The `e` parameter is the actual `SAXParseException` that has occurred. A conforming `Parser` continues to parse the current XML content to the end of the document, having returned from reporting a warning. An implementation of this method may throw another `SAXException`, possibly wrapping this or a new `SAXParseException`. The parsing behavior of a `Parser` when another `SAXException` is raised is implementation dependent. A conforming Parser shall not permit infinite recursion to occur by repeatedly delivering re-thrown `SAXException` notifications through this method.

`void fatalError( SAXParseException e )`	The `Parser` invokes this method to notify the client that a fatal error has occurred during parsing. Such an error is defined in section 1.2 of the W3C XML specification 1.0. The application shall assume that the XML content being parsed by the `Parser` is unusable after the `Parser` has invoked this method and may continue only to collect any additional error messages generated. After reporting a fatal error, a conforming `Parser` may continue to parse the XML content to EOF (if able) or immediately terminate parsing. The `e` parameter is the actual `SAXParseException` that has occurred. An implementation of this method may throw another `SAXException`, possibly wrapping this or a new `SAXParseException`. The parsing behavior of a `Parser` when another such `SAXException` is raised is implementation dependent. A conforming Parser shall not permit infinite recursion to occur by repeatedly delivering re-thrown `SAXException` notifications through this method.

`void error( SAXParseException e )throws SAXException`	The `Parser` invokes this method to notify the client that a recoverable error has occurred during parsing. Such an error is defined in section 1.2 of the W3C XML specification 1.0. The default behavior of this method is to take no action. (In some circumstances, the error may be conveyed to a human user via some error messaging facility.) The `e` parameter is the actual `SAXParseException` that has occurred. A conforming `Parser` will continue to parse the current XML content to the end of the document, having returned from reporting an error. If the `Parser` is unable to do so, it shall report a fatal error, although the XML 1.0 specification does not require it to do so. An implementation of this method may throw another `SAXException`, possibly wrapping this or a new `SAXParseException`. The parsing behavior of a `Parser` when another such `SAXException` is raised is implementation dependent. A conforming Parser shall not permit infinite recursion to occur by repeatedly delivering re-thrown `SAXException` notifications through this method.

2.2.6 org.xml.sax.HandlerBase

Description

The `org.xml.sax.Handlerbase` is a convenience class provided by this package to aid a developer creating `DocumentHandler`, `DTDHandler`, `ErrorHandler` and `EntityResolver` implementations for use/registration with a `Parser`, through subclassing and overriding of the appropriate interface methods.

Using this class is *not* required to implement any of the SAX interfaces mentioned here.

Methods

InputSource resolveEntity(String publicId, String systemId)	See `org.xml.sax.EntityResolver` for details. The default implementation returns `null`.
void notationDecl(String name, String publicId, String systemId)	See `org.xml.sax.DTDHandler` for details. The default implementation is to do nothing.
void unparsedEntityDecl(String name, String publicId, String systemId, String notationName)	See `org.xml.sax.DTDHandler` for details. The default implementation is to do nothing.
void setDocumentLocator(Locator l)	See `org.xml.sax.DocumentHandler` for details. The default implementation is to do nothing.
void startDocument()	See `org.xml.sax.DocumentHandler` for details. The default implementation is to do nothing.
void endDocument()	See `org.xml.sax.DocumentHandler` for details. The default implementation is to do nothing.
void startElement(String name, AttributeList atts)	See `org.xml.sax.DocumentHandler` for details. The default implementation is to do nothing.
void endElement(String name)	See `org.xml.sax.DocumentHandler` for details. The default implementation is to do nothing.

void characters(char[] ch, int start, int length)	See `org.xml.sax.DocumentHandler` for details. The default implementation is to do nothing.
void ignorableWhitespace(char[] ch, int start, int length)	See `org.xml.sax.DocumentHandler` for details. The default implementation is to do nothing.
void processingInstruction(String target, String data)	See `org.xml.sax.DocumentHandler` for details. The default implementation is to do nothing.
void warning(SAXParseException e)throws SAXException	See `org.xml.sax.ErrorHandler` for details. The default implementation is to do nothing.
void error(SAXParseException e)throws SAXException	See `org.xml.sax.ErrorHandler` for details. The default implementation is to do nothing.
void fatalError(SAXParseException e)throws SAXException	See `org.xml.sax.ErrorHandler` for details. The default implementation is to do nothing.

2.2.7 org.xml.sax.InputSource

Description

The `org.xml.sax.InputSource` class encapsulates information about an input source of XML content for a `Parser` into a single object representation.

A `Parser` uses an `InputSource` to obtain information about XML content, including how to consume the content. The `Parser` will first use the character stream (`Reader`) if it is available. If not, it uses a bytestream (`InputStream`). If neither is available, the `Parser` attempts to resolve and open a connection to the URI specified by the system identifier.

The information provided includes:

- a public identifier
- a system identifier
- a `java.io.InputStream` (with optional specified encoding) bytestream, and/or a `java.io.Reader` character stream.

An `InputSource` is used in two places:

- as an input parameter to `Parser.parse()` method(s)
- as a return value from the `EntityResolver.resolveEntity()` method

A `Parser` may not modify the content of an `InputSource`.

`InputSource` instances passed to the `Parser.parse()` method(s) shall be associated with a properly resolved system identifier so that a Parser may resolve any relative URIs in a document. If a system identifier is not provided, parsing errors will result when external entities specified by relative URIs are parsed. This is due to these relative URIs being unresolvable.

Methods

`InputSource()`	construct an `InputSource`.
`InputSource( String systemId )`	construct an `InputSource` with a specified `systemId`.
`InputSource( InputStream s )`	construct an `InputSource` with the specified `InputStream s`.
`InputSource( Reader r )`	construct an `InputSource` with the specified `Reader r`.
`void setPublicId( String publicId )`	set the public id of the `InputSource`. This is optional but strongly encouraged to provide a public id.

`String getPublicId()`	return the public id of the `InputSource` or `null`.
`void setSystemId( String systemId )`	set the system id of the `InputSource`. Applications are required to set a system id in order to provide a base URI for resolving relative URIs found in the encapsulated document.
`String getSystemId()`	return the system id of the `InputSource` or `null`.
`setByteStream( InputStream is )`	set the `InputStream` to be associated with the `InputSource`. A `Parser` ignores this value if a character stream (`Reader`) is also associated with this `InputSource`.
`InputStream getByteStream()`	return the current `InputStream`.
`void setEncoding( String encoding )`	set the character encoding for the current `InputStream` or system id. This encoding must conform to the encoding declaration rules in the XML specification (1.0) section 4.3.3. A conforming `Parser` ignores this value if a character stream is also specified for the `InputSource`.
`String getEncoding()`	get the character encoding for the current `InputStream`, or URI, or `null` if none is specified.
`void setCharacterStream( Reader cs )`	set the character stream (`Reader`) for the `InputSource`. If a `Reader` is set, then a conforming Parser shall use it to obtain content, ignoring any `InputStream` and not attempting to resolve any system id. The character stream (`Reader`) shall not include a byte order mark.
`Reader getCharacterStream()`	return the character stream (`Reader`) for the `InputSource`.

2.2.8 org.xml.sax.Locator

Description

A conforming Parser supplies the `org.xml.sax.Locator` interface and registers it (using its `setDocumentLocator()` method) with the `DocumentHandler` it is associated with (using the `Parser.setDocumentHandler()` method) before beginning a parse operation on a particular `InputSource`.

The `Locator` instance, supplied by the `Parser`, enables the `DocumentHandler` to obtain information about the location of any `DocumentHandler` event for the duration of that event/notification. Applications shall not invoke a `Locator` outside a `DocumentHandler` event/notification method, as these values have undefined and unpredictable results.

Methods

`String getPublicId()`	return the public identifier for the current `DocumentHandler` event, or `null` if none is available.
`String getSystemId()`	return the system identifier for the current `DocumentHandler` event, or `null` if none is available. If the identifier is a URI then the `Parser` shall resolve it, returning a fully resolved URI.
`int getLineNumber()`	return the line number where the current `DocumentHandler` event ends, or -1 if none is available. Note that this line number is associated with the line position of the first character after the text that is the subject of the current event.
`int getColumnNumber()`	return the column number where the current `DocumentHandler` event ends, or -1 if none is available. Note that this column number is associated with the column position of the first character after the text that is the subject of the current event.

2.2.9 org.xml.sax.Parser

Description

The `org.xml.sax.Parser` is an interface implemented by conforming parsers. It enables application code to register handlers to receive notification of a variety of XML document parsing related events, and to initiate parsing XML content from a character stream (`Reader`), bytestream (`InputStream`) or URI (`String`) described by an `InputSource`.

A conforming `Parser` implementation shall implement a public, no-args constructor. (Other constructor signatures are permitted, but not used or required by this specification.)

A particular `Parser` instance is reusable but not reentrant. An application may reuse a Parser instance (possibly with a different or the same `InputSource`) after a previous invocation of a parse() method has completed. However, an application may not invoke a parse operation recursively or simultaneously while that instance is already embarked upon a parsing operation.

Methods

`void setLocale( Locale l )throws SAXException`	set the `Locale` for the `Parser` to use to report any errors and/or warnings. A conforming `Parser` is not required to support localization of warnings and/or errors. However if a `Parser` cannot support a specified `Locale`, it must raise an appropriate `SAXException`. It is an error to change the `Locale` while in the process of parsing an `InputSource`.
`void setEntityResolver( EntityResolver er )`	set a custom `EntityResolver` for the `Parser`, or `null`. If no `EntityResolver` is specified, the `Parser` shall resolve entities itself. The `EntityResolver` may be specified at any time, including during an ongoing parsing operation. The results of the set take effect immediately; the `Parser` will use the new value immediately after returning from this method.
`void setDTDHandler( DTDHandler dtdh )`	set a custom `DTDHandler` for the `Parser` or `null`. If no `DTDHandler` is specified, no DTD related events are generated. The `DTDHandler` may be specified at any time, including during an ongoing parsing operation. The results of the set take immediate effect; the `Parser` will use the new value immediately after returning from this method.

`void setErrorHandler( ErrorHandler eh )`	set a custom `ErrorHandler` for the `Parser` or `null`. If no `ErrorHandler` is specified, no errors and warnings are generated, except fatal errors. A fatal error will result in `SAXException` being thrown. The `ErrorHandler` may be specified at any time, including during an ongoing parsing operation. The results of the set take immediate effect; the `Parser` will use the new value immediately after returning from this method.

`void parse( InputSource is ) throws IOException, SAXException`	direct the `Parser` to parse the XML content described by the `InputSource`. The appropriate `DocumentHandler`, `DTDHandler`, `ErrorHandler`, and `EntityResolver` method(s) shall be called (in order) as specified. This method may not be called recursively, or simultaneously on the same instance. This method throws `SAXException` to indicate that an error has occurred during parsing that was not handled by the `ErrorHandler,` if any. It throws `IOException` if an IO error condition occurs on the underlying `InputStream`, `Reader` or other URI connection.
`void parse( String systemId ) throws IOException, SAXException`	direct the `Parser` to parse the XML content described by the `URI system identifier`. The appropriate `DocumentHandler`, `DTDHandler`, `ErrorHandler`, and `EntityResolver` method(s) shall be called (in order) as specified. This method may not be called recursively, or simultaneously on the same instance. This method throws `SAXException` to indicate an error has occurred during parsing that was not handled by the `ErrorHandler,` if any (or ignored). It throws `IOException` if an IO error condition occurred on the underlying URI connection.

void setDocumentHandler(DocumentHandler dh)	set a custom `DocumentHandler` for the `Parser` or `null`. If no `DocumentHandler` is specified, no Document-related events are generated. The `DocumentHandler` may be specified at any time, including during an ongoing parsing operation. The results of the set take immediate effect; the `Parser` will use the new value immediately after returning from this method.

2.2.10 org.xml.sax.SAXException

Description

The `org.xml.sax.SAXException` class extends `java.lang.Exception` and is provided to describe basic warning and error information for either a `Parser` or application code written in conjunction with a `Parser`.

A `Parser` implementor or application writer may subclass and extend the basic capabilities to provide additional functionality as required.

Any `SAXException` instances initialized by a `Parser` supporting localization of error and warning messages shall set any messages in the appropriate (current) `Locale` as set upon the originating `Parser` using its `setLocale()` method.

Methods

SAXException(String msg)	construct a SAXException with `msg String`.
SAXException(Exception e)	construct a `SAXException` to encapsulate `Exception e`.
SAXException(Exception e, String msg)	construct a `SAXException` to encapsulate `Exception e`, with `msg String`.
String getMessage()	return the message `String` associated with this `SAXException` or `null`. If no message was set on the `SAXException`, then this message shall return the message (if any) associated with an encapsulated `Exception`.
Exception getException()	return the encapsulated `Exception` associated with this `SAXException` or `null`.

2.2.11 org.xml.sax.SAXParseException

Description

The `org.xml.sax.SAXParseException` class extends `SAXException` and is provided to describe basic warning and error information, including error location information within the content initialized from a `Locator`, for either a `Parser` or application code written in conjunction with a `Parser`.

A `Parser` implementor or application writer may subclass and extend the basic capabilities herein to provide additional functionality as required.

Any `SAXParseException` instances initialized by a `Parser` supporting localization of error and warning messages shall set any messages in the appropriate (current) `Locale` as set upon the originating `Parser` via its `setLocale()` method.

Methods

SAXParseException(String msg, Locator l)	construct a SAXParseException with msg String and a Locator l. Note that this constructor may only be used within the context of a DocumentHandler event/notification method because it uses a Locator.
SAXParseException(String msg, Locator l, Exception e)	construct a SAXParseException with msg String, a Locator l, and encapsulating an Exception e. Note that this constructor may only be used within the context of a DocumentHandler event/notification method because it uses a Locator.
SAXParseException(String msg, String publicId, String systemId, int lineNumber, int columnNumber, Exception e)	construct a SAXParseException, with msg String, a (String) public identifier (or null), a (String) system identifier (a fully resolved URI), a line number (or -1), a column number (or -1), and an Exception e (or null).
String getPublicId()	return the public identifier of the XML content that is the subject of the exception. see Locator for details.
String getSystemId()	return the system identifier of the XML content that is the subject of the exception. see Locator for details.
int getLineNumber()	return the line number within the content that is the subject of the exception, or -1 if none is available. See Locator definition for details.
int getColumnNumber()	return the column number within the content that is the subject of the exception or -1 if none is available. See Locator for details.

CHAPTER 3

DOM

3.1 Overview

The Document Object Model is a World Wide Web Consortium standard for representing a parsed XML “document” as a logical tree structure and exposing that structure programmatically.

This specification subsumes the semantics for Core DOM level 1 as described in the W3C specification and the Java language binding in Appendix D therein.

This specification does not specify DOM (HTML) Level 1 support; that is deferred to a later version of this specification.

Note that the W3C specifications do not provide a complete API. This specification provides additional API in Chapter 4.

3.2 API Definition(s)

3.2.1 org.w3c.dom.Attr

Description

The `org.w3c.dom.Attr` interface extends `org.w3c.dom.Node`.

It is used to represent an attribute name/value associated with an `Element`. The allowable values for an attribute are typically defined in a DTD.

`Attr` objects inherit the `Node` interface, but because they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree. As a result, the `Node` attributes parentNode, previousSibling, and nextSibling have a `null` value for `Attr` objects. The DOM takes the view that attributes are properties of elements rather than having an identify separate from the elements they are associated with. This should make it more efficient to implement features such as default attributes associated with all elements of a given type.

`Attr` nodes may not be immediate children of a `DocumentFragment`. However, they can be associated with `Element` nodes contained within a `DocumentFragment`. In short, users and implementors of the DOM need to be aware that `Attr` nodes have some things in common with other objects inheriting the `Node` interface, but they also are quite distinct.

The attribute's effective value is determined as follows:

- If this attribute has been explicitly assigned any value, that value is the attribute's effective value.
- If there is a declaration for this attribute and that declaration includes a default value, then that default value is the attribute's effective value.
- Otherwise, the attribute does not exist on this element in the structure model until it has been explicitly added.

Note that the nodeValue attribute on the `Attr` instance can also retrieve the string version of the attribute's value(s).

In XML, where the value of an attribute can contain entity references, the child nodes of the `Attr` node provide a representation in which entity references are not expanded. These child nodes may be either `Text` or `EntityReference` nodes. Because the attribute type may be unknown, there are no tokenized attribute values.

Methods

`String getName()`	return the value of the `name` property
`boolean getSpecified()`	return the value of the `specified` property This is true if this attribute was explicitly given a value in the original document; otherwise, it is false. Note that the implementation is in charge of this attribute, not the user. If the user changes the value of the attribute (even if it ends up having the same value as the default value) then the specified flag is automatically flipped to true. To re-specify the attribute as the default value from the DTD, the user must delete the attribute. The implementation will then make a new attribute available with `specified` set to false and the default value (if one exists).
`void setValue( String value )`	set the value of the `value` property. This creates a `Text` node containing unparsed content.
`String getValue()`	return the value of the `value` property. Character and entity references are resolved.

3.2.2 org.w3c.dom.CDATASection

Description

The `org.w3c.dom.CDATASection` is an interface that extends `org.w3c.dom.Text.`

CDATA sections are used to escape blocks of text containing characters that would otherwise be regarded as markup. The only delimiter recognized in a CDATA section is the "]]>" string that ends the CDATA section. CDATA sections cannot be nested. Their primary purpose is including material such as XML fragments, without needing to escape all the delimiters.

The data attribute of the `Text` node holds the text that is contained by the CDATA section. Note that this may contain characters that need to be escaped outside of CDATA sections. Depending on the character encoding ("charset") chosen for serialization, it may be impossible to write out some characters as part of a CDATA section.

The `CDATASection` interface inherits the `CharacterData` interface through the `Text` interface.

Note that adjacent `CDATASection` nodes are not merged by use of the `Element.normalize()` method.

Methods

This interface does not define any additional methods or constants.

3.2.3 org.w3c.dom.CharacterData

Description

The `org.w3c.dom.CharacterData` interface extends frm `org.w3c.dom.Node` with a set of attributes and methods for accessing character data in the DOM. For clarity, this set is defined here rather than on each object that uses these attributes and methods. No DOM objects correspond directly to `CharacterData`, although `Text` and others do inherit the interface from it. All offsets in this interface start from 0.

Methods

`void setData( String data ) throws DOMException`	set the value of the `data` property. This method may raise the `DOMException` `NO_MODIFICATION_ALLOWED_ERR` if the target `Node` is read only.
`String getData() throws DOMException`	returns the value of the `data` property. This method may raise the `DOMException` `DOMSTRING_SIZE_ERR` when the length of the data to be returned exceeds the platform's capability to represent it as a `String`.
`int getLength()`	returns the value of the `length` property.
`String substringData( int offset, int count ) throws DOMException;`	return a substring of the data property from the index `offset` within the property for `count` characters. If `offset+count > length` then `count` is adjusted to count = `length-offset` This method may raise the following `DOMException`(s): • `INDEX_SIZE_ERR` if the specified `offset` is negative, or greater than the number of characters in the data. • `INDEX_SIZE_ERR` if the count is negative. • `DOMSTRING_SIZE_ERR` if the substring requested is larger than the maximum size of a `String`.
`void appendData( String arg ) throws DOMException`	append the specified `String arg` to the end of the current `data` property value. This method may raise the `DOMException` `NO_MODIFICATION_ALLOWED_ERR` if the target `Node` is read only.

void insertData(int offset, String arg) throws DOMException	insert the specified `String arg` at the specified `offset`. This method may raise the following `DOMException`(s): • `INDEX_SIZE_ERR` if the `offset` is either negative or exceeds the number of characters in the data • `NO_MODIFICATION_ALLOWED_ERR` if the target node is read only.
void deleteData(int offset, int count) throws DOMException	delete `count` characters at `offset` in the data. If `offset+count > length` then `count` is adjusted to `count = length-offset`. This method may raise the following `DOMException`(s): • `INDEX_SIZE_ERR` if the specified `offset` is negative or greater than the number of characters in the data. • `INDEX_SIZE_ERR` if the count is negative.
void replaceData(int offset, int count, String arg) throws DOMException	replace the (sub)string of length `count`, in `data` at `offset` with `arg`. If `offset+count > length` then `count` is adjusted to `count = length-offset`. This method may raise the following `DOMException`(s): • `INDEX_SIZE_ERR` if the specified `offset` is negative or greater than the number of characters in the data. • `INDEX_SIZE_ERR` if the count is negative. • `NO_MODIFICATION_ALLOWED_ERR` if the target node is read only.

3.2.4 org.w3c.dom.Comment

Description

The `org.w3c.dom.Comment` is an interface that extends `org.w3c.dom.CharacaterData`.

This represents the content of a comment, i.e., all the characters between the starting ’<!--’ and ending ’-->’. Note that this is the definition of a comment in XML, and, in practice, HTML, although some HTML tools may implement the full SGML comment structure.

Methods

This interface does not define any additional methods or constants.

3.2.5 org.w3c.dom.DOMException

Description

The `org.w3c.dom.DOMException` is a class that extends `java.lang.RuntimeException`.

DOM operations only raise exceptions in "exceptional" circumstances, i.e., when an operation is impossible to perform, either for logical reasons, because data is lost, or because the implementation has become unstable. In general, DOM methods return specific error values in ordinary processing situation, such as out-of-bound errors when using `NodeList`.

Implementations may raise exceptions under other circumstances. For example, implementations may raise an implementation-dependent exception if a `null` argument is passed.

Constants

short INDEX_SIZE_ERR	exception code if an index is negative or exceeds bounds.
short DOMSTRING_SIZE_ERR	exception code if target range of text cannot be represented as a `String`.
short HIERARCHY_REQUEST_ERR	exception code if target `Node` insert is attempted at an illegal location in a `Document` hierarchy.
short WRONG_DOCUMENT_ERR	exception code if target `Node` is used in the context of a `Document` other than the one that created it.
short INVALID_CHARACTER_ERR	exception code if an invalid character is specified.
short NO_DATA_ALLOW_ERR	exception code if data is specified for a target `Node` for which no data is allowed.
short NO_MODIFICATION_ALLOWED_ERR	exception code if an attempt is made to modify a target `Node` that does not permit modifications (read only).
short NOT_FOUND_ERR	exception code if an attempt was made to reference a `Node` in a context where it does not exist.
short NOT_SUPPORTED_ERR	exception code if the implementation does not support the type of object or operation requested.
short INSUE_ATTRIBUTE_ERR	exception code if an attempt is made to add an attribute that is already in use elsewhere in context.

Methods

DOMException(short code, String msg)	construct a `DOMException` with the specified exception `code` and a `msg` String (optionally `null`).

Fields

short code	one of the exception codes above.

3.2.6 org.w3c.dom.DOMImplementation

Description

The `org.w3c.dom.DOMImplementation` interface provides a number of methods for performing operations that are independent of any particular instance of the document object model.

Methods

`boolean hasFeature( String feature, String version )`	return `true` if the underlying implementation supports the requested `feature` at the specified `version`, otherwise `false`. Valid feature names are: • "`XML`" • "HTML" Valid version is: "`1.0`" In this version of the specification, only the "`XML`" feature is supported.

3.2.7 org.w3c.dom.Document

Description

The `org.w3c.dom.Document` is an interface that extends `org.w3c.dom.Node`.

A `Document` represents an entire instance of an XML document. Conceptually, it is the "root" of a particular document "tree".

Since `Element`(s), `Node`(s), `Comment`(s), `Text`, and `ProcessingInstruction`(s) cannot exist outside the context of a Document, this interface provides factory methods to create these.

Methods

DocumentType getDocumentType()	return the `DocumentType` of this `Document`, or `null`.
DOMImplementation getDOMImplementation()	return the `DOMImplementation` for this `Document`.
Element getDocumentElement()	return the “root” element of this `Document`.
Element createElement(String elementName) throws DOMException	create an `Element` with the specified name. Note that no DTD validation occurs for the `elementName`. This method may raise a `DOMException` of type `INVALID_CHAR_ERR` if the specified name contains an invalid character.
DocumentFragment createDocumentFragment()	create an empty `DocumentFragment`.
Text createTextNode(String data)	create a `Text Node` containing the specified `String`.
Comment createComment(String data)	create a `Comment` with the specified `String`.
CDATASection createCDATASection(String data) throws DOMException	create a `CDATASection` containing the specified `String`. This method may raise a `DOMException` with type `NOT_SUPPORTED_ERR` if the document is not an XML document.

ProcessingInstruction createProcessingInstruction (String target, String data) throws DOMExcepton	create a `ProcessingInstruction` with the specified `target` and `data`. This method may raise the following DOMException(s): • `INVALID_CHAR_ERR` if the target contains an invalid character • `NOT_SUPPORTED_ERR` if the document is not an XML document
Attr createAtrribute(String name) throws DOMException	create an `Attr` with the specified `name`. This method may raise a `DOMException` with type `INVALID_CHAR_ERR` if the specified name contains an invalid character.
EntityReference createEntityReference(String name) throws DOMException	create an `EntityReference` with the specified `name`. This method may raise the following DOMException(s): • `INVALID_CHAR_ERR` if the name contains an invalid character • `NOT_SUPPORTED_ERR` if the document is not an XML document
NodeList getElementsByTagName(String tagname)	return a `NodeList` (possibly empty) matching the specified `tagName` in document traversal order.

3.2.8 org.w3c.dom.DocumentFragment

Description

The `org.w3c.dom.DocumentFragment` interface extends the `org.w3c.dom.Node` interface.

`DocumentFragment` is a "lightweight" or "minimal" `Document` object used to extract a portion of a document's tree or to create a new fragment of a document. Imagine implementing a user command like cut or rearranging a document by moving fragments

around. It is desirable to have an object which can hold such fragments and it is quite natural to use a Node for this purpose. While it a `Document` object could fulfil this role, it can potentially be a heavyweight object, depending on the underlying implementation. What is needed is a lightweight object. `DocumentFragment` is such an object.

Various operations, such as inserting nodes as children of another `Node`, may take `DocumentFragment` objects as arguments. This results in all the child nodes of the `DocumentFragment` being moved to the child list of this node.

The children of a `DocumentFragment` node are zero or more nodes representing the tops of any sub-trees defining the structure of the document.

`DocumentFragment` nodes do not need to be well-formed XML documents, although they do need to follow the rules imposed upon well-formed XML parsed entities, which can have multiple top nodes. For example, a `DocumentFragment` might have only one child and that child node could be a `Text` node. Such a structure model represents neither an HTML document nor a well-formed XML document.

When a `DocumentFragment` is inserted into a `Document` (or any other `Node` that may take children), the *children* of the `DocumentFragment` are inserted into the `Node,` not the `DocumentFragment` itself. The `DocumentFragment` is useful when the user wishes to create nodes that are siblings; the `DocumentFragment` acts as the parent of these nodes so that the user can use the standard methods from the `Node` interface, such as `insertBefore()` and `appendChild()`.

Methods

This interface does not define any additional methods.

3.2.9 org.w3c.dom.DocumentType

Description

The `org.w3c.dom.DocumentType` is an interface.

Each `Document` has a doctype attribute whose value is either `null` or a `DocumentType` object. The `DocumentType` interface in the DOM Level 1 Core provides an interface to the list of entities that are defined for the document.

Methods

`String getName()`	return the name of the document's DTD.
`NameNodeMap getEntities()`	return a `NamedNodeMap` that enumerates the internal and external entities declared in the document's DTD. Any duplicates are discarded.
`NameNodeMap getNotations()`	return a NamedNodeMap that enumerates the notations declared in the document's DTD. Any duplicates are discarded.

3.2.10 org.w3c.dom.Element

Description

The `org.w3c.dom.Element` is an interface that extends `org.w3c.dom.Node`.

An `Element` represents an XML element construct in a `Document`.

An `Element` may have attributes associated with it. Because the `Element` interface inherits from `Node`, the generic `Node` interface method `getAttributes()` may be used to retrieve the set of all attributes for an element. The `Element` interface has methods to retrieve either an `Attr` object or value by name. In XML, where an attribute value may contain entity references, an `Attr` object should be retrieved to examine the possibly complex subtree representing the attribute value.

Methods

`String getTagName()`	return the name of the element. For XML documents, case is preserved.
`void setAttribute( String name, String value) throws DOMException`	add a new attribute. If an attribute with that name is already present in the element, its value is changed to that of the value parameter. The value is a simple string; it is not parsed as it is being set. Any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written. To assign an attribute value that contains entity references, the user must create an `Attr` node plus any `Text` and `EntityReference` nodes, build the appropriate subtree, and use `setAttributeNode` to assign it as the value of an attribute. This method may raise the following DOMException(s); • `INVALID_CHARACTER_ERR` if the specified name contains an invalid character. • `NO_MODIFICATION_ALLOWED_ERR` if `Element` is read only.
`String getAttribute( String name )`	Return the value of the named attribute, or `null` if the attribute is not specified and has no default value predefined.

`Attr setAttributeNode( Attr newAttr ) throws DOMException`	add a new attribute. If an attribute with the same name already exists, its value is replaced with the new one. This returns the previous attribute value (if any) or `null`. This method may raise the following DOMException(s): • `WRONG_DOCUMENT_ERR` if the new attribute was created from a different `Document` than the one that created the `Element`. • `NO_MODIFICATION_ALLOWED_ERR` if `Element` is read only. • `INUSE_ATTRIBUTE_ERR` if the specified attribute is already an attribute of another `Element`. An attribute shall be cloned explicitly to be re-used in other `Element`(s).
`Attr getAttributeNode( string name )`	return the specified attribute, or `null` if no such attribute exists.

Attr removeAttributeNode(Attr oldAttr) throws DOMException	remove the specified attribute. This method may raise the following DOMException(s): • NOT_FOUND_ERR if the specified attribute is not a member of the Element. • NO_MODIFICATION_ALLOWED_ERR if Element is read only.
NodeList getElementsByName(String name)	return a NodeList of all descendant elements with a given tag name, in the order in which they would be encountered in a preorder traversal of the Element tree. The string "*" matches all names.
void normalize()	transform all Text nodes in the full depth of the subtree underneath this Element into a "normal" form where only markup (e.g., tags, comments, processing instructions, CDATA sections, and entity references) separates Text nodes. (There are no adjacent Text nodes.) This method can be used to ensure that the DOM view of a document is the same as if it were saved and re-loaded, and is useful when using operations such as XPointer lookups that depend on a particular document tree structure.

3.2.11 org.w3c.dom.Entity

Description

The org.w3c.dom.Entity interface extends org.w3c.dom.Node.

This interface represents an entity, either parsed or unparsed, in an XML document. Note that this models the entity itself, not the entity declaration. Entity declaration modeling has been left for a later level of the DOM specification.

The nodeName attribute inherited from Node contains the name of the entity.

An XML processor may choose to expand entities completely before passing the structure model to the DOM; in this case there will be no EntityReference nodes in the document tree.

XML does not mandate that a non-validating XML processor read and process entity declarations made in the external subset or declared in external parameter entities. This means that parsed entities declared in the external subset need not be expanded by some classes of applications, and that the replacement value of the entity may not be available.

When the replacement value is available, the corresponding `Entity` node's child list represents the structure of that replacement text. Otherwise, the child list is empty.

The resolution of the children of the `Entity` (the replacement value) may be evaluated lazily; actions by the user (such as calling the childNodes method on the `Entity Node`) are assumed to trigger the evaluation.

The DOM Level 1 does not support editing `Entity` nodes. If a user wants to make changes to the contents of an `Entity`, every related `EntityReference` node has to be replaced in the structure model by a clone of the `Entity`'s contents, and then the desired changes must be made to each of those clones instead. All the descendants of an `Entity` node are read only.

An `Entity` node does not have any parent.

Methods

`String getPublicId()`	return the public identifier (if any specified) or `null`.
`String getSystemId()`	return the system identifier (if any specified) or `null`.
`String getNotationName()`	return the notation for the Entity, this is `null` for parsed entities.

3.2.12 org.w3c.dom.EntityReference

Description

The `org.w3c.dom.EntityReference` interface extends `org.w3c.dom.Node`.

`EntityReference` objects may be inserted into the structure model when an entity reference is in the source document, or when the user wishes to insert an entity reference.

Note that character references and references to predefined entities are considered to be expanded by the XML processor so that characters are represented by their Unicode equivalent rather than by an entity reference.

The XML processor may completely expand references to entities while building the structure model, instead of providing `EntityReference` objects. If it does provide such objects, then for a given `EntityReference` node, there may be no `Entity` node representing the referenced entity. If such an `Entity` exists, then the child list of the `EntityReference` node is the same as that of the `Entity` node. As with the `Entity` node, all descendants of the `EntityReference` are read only.

The resolution of the children of the `EntityReference` (the replacement value of the referenced `Entity`) may be evaluated lazily; actions by the user (such as calling the childNodes method on the `EntityReference` node) are assumed to trigger the evaluation.

Methods

This interface defines no additional methods.

3.2.13 org.w3c.dom.NamedNodeMap

Description

The `org.w3c.dom.NamedNodeMap` interface is used to represent collections of nodes that can be accessed by name. Note that `NamedNodeMap` does not inherit from `NodeList`; `NamedNodeMaps` are not maintained in any particular order. Objects contained in an object implementing `NamedNodeMap` may also be accessed by an ordinal index. This is simply to allow convenient enumeration of the contents of a `NamedNodeMap`, and does not imply that the DOM specifies an order to these `Node`s.

Methods

`Node getNamedItem( String name )`	return the `Node` with the specified name, or null if not present in the `NamedNodeMap`.
`Node setNamedItem( Node arg ) throws DOMException`	add a `Node` to the `NamedNodeMap` with the specified name. This returns the preexisting `Node` (if any) this the new `Node` replaces, or `null`. This method may raise the following DOMException(s): • `WRONG_DOCUMENT_ERR` if the new attribute was created from a different `Document` than the one that created the `Element`. • `NO_MODIFICATION_ALLOWED_ERR` if `NamedNodeMap` is read only. • `INUSE_ATTRIBUTE_ERR` if the specified attribute is already an attribute of another `Element`. An attribute shall be cloned explicitly in order to be re-used in other `Elements`. • `HIERARCHY_REQUEST_ERR` when an attempt is made to add an `Element` node to a `NamedNodeMap` associated with an `Attr` list.
`Node removeNamedItem( String name ) throws DOMException`	remove the `Node` with the specified name. If the removed `Node` is an `Attr` with a default value, it is immediately replaced. This returns the `Node` removed. This method may raise the DOMException `NOT_FOUND_ERR` if there is no `Node` with the name in the map.
`Node item(int index)`	return the `Node` at the specified index, or `null` if out of bounds.
`int getLength()`	return the value of the length property; the number of `Node` items in the map.

3.2.14 org.w3c.dom.Node

Description

The `org.w3c.dom.Node` interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the `Node` interface expose methods for dealing with children, not all objects implementing the `Node` interface may have children. For example, `Text` nodes may not have children; adding children to such nodes raises a DOMException.

The attributes `nodeName`, `nodeValue` and `attributes` are included as a mechanism to access node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific nodeType (e.g., nodeValue for an `Element` or attributes for a `Comment`), this returns `null`. Note that the specialized interfaces may contain additional and more convenient mechanisms to retrieve and set the relevant information.

The values of nodeName, nodeValue, and attributes vary according to the node type:

Node Type	nodeName	nodeValue	attributes
`Element`	tagName	null	`NamedNodeMap`
`Attr`	name of Attr	attribute value	`null`
`Text`	#text	text content	`null`
`CDATASection`	#cdata-section	CDATA content	`null`
`EntityReference`	entity name	`null`	`null`
`Entity`	entity name	`null`	`null`
`ProcessingInstruction`	target	content	`null`
`Comment`	#comment	comment	`null`
`Document`	#document	`null`	`null`
`DocumentType`	doc type name	`null`	`null`
`DocumentFragment`	#document-fragment	`null`	`null`
`Notation`	notation name	`null`	`null`

Constants

short ELEMENT_NODE	node type for `Element`
short ATTRIBUTE_NODE	node type for `Attr`
short TEXT_NODE	node type for `Text`
short CDATA_SECTION_NODE	node type for `CDATASection`
short ENTITY_REFERENCE_NODE	node type for `EntityReference`
short ENTITY_NODE	node type for `Entity`
short PROCESSING_INSTRUCTION_NODE	node type for `ProcessingInstruction`
short COMMENT_NODE	node type for `Comment`
short DOCUMENT_NODE	node type for `Document`
short DOCUMENT_TYPE_NODE	node type for `DocumentType`
short DOCUMENT_FRAGMENT_NODE	node type for `DocumentFragment`
short NOTATION_NODE	node type for `Notation`

Methods

`String getNodeName()`	return the name of the `Node`. This value is `Node` (sub)interface-dependent.
`void setNodeValue( String nodeValue ) throws DOMException`	set the `Node` value. The value is `Node` (sub)interface-dependent. This method may raise a DOMException `NO_MODIFICATION_ALLOWED_ERR` when the `Node` is read only.
`String getNodeValue()`	return the value of the `Node`.
`short getNodeType()`	return the `Node` type (see constants above).
`Node getNodeParent()`	return the parent `Node` of this `Node`, or `null`.
`NodeList getChildNodes()`	return a `NodeList` (possibly empty) of the immediate child `Node`(s) of this `Node`.
`Node getFirstChild()`	get the first child of this `Node`, or `null`.
`Node getLastChild()`	get the last child of this `Node`, or `null`.
`Node getPreviousSibling()`	return the previous sibling of this `Node`, or `null`.
`Node getNextSibling()`	return the next sibling of this `Node`, or `null`.
`NamedNodeMap getAttributes()`	return a `NamedNodeMap` of the attributes for this `Node`.
`Document getOwnerDocument()`	return the `Document` this `Node` belongs to/was created by.

Node insertBefore(Node newChild, Node refChild) throws DOMException	insert the node `newChild` before the existing child node `refChild`. If `refChild` is `null`, insert `newChild` at the end of the list of children. If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If the `newChild` is already in the tree, it is first removed. This returns the `Node` being inserted. This method may raise the following `DOMException`(s): • `WRONG_DOCUMENT_ERR` if the `newChild` was created from a different `Document` than the one that created the `Node`. • `NO_MODIFICATION_ALLOWED_ERR` if `Node` is read only. • `NOT_FOUND_ERR` if the `refChild` is not a child of this `Node`. • `HIERARCHY_REQUEST_ERR` if the `Node` is of a type that does not allow children of the type of the `newChild`, or the `Node` is an ancestor of this one.

Node replaceChild(Node newChild, Node oldChild) throws DOMException	replace the `oldChild` with the `newChild`. This returns the `Node` being replaced. This method may raise the following `DOMException`(s): • `WRONG_DOCUMENT_ERR` if the `newChild` was created from a different `Document` than the one that created the `Node`. • `NO_MODIFICATION_ALLOWED_ERR` if `Node` is read only. • `NOT_FOUND_ERR` if the `refChild` is not a child of this `Node`. • `HIERARCHY_REQUEST_ERR` if the `Node` is of a type that does not allow children of the type of the `newChild`, or the `Node` is an ancestor of this one.
Node removeChild(Node oldChild) throws DOMException	remove the child specified. This returns the child being removed. This method may raise the following `DOMException`(s): • `NO_MODIFICATION_ALLOWED_ERR` if `Node` is read only. • `NOT_FOUND_ERR` if the `refChild` is not a child of this `Node`.

`Node appendChild( Node newChild ) throws DOMException`	append the child specified to the end of the list of children. If the child specified is already in the list of children, it is first deleted, then appended. This returns the child appended. This method may raise a `DOMException`(s): • `WRONG_DOCUMENT_ERR` if the `newChild` was created from a different `Document` than the one that created the `Node`. • `NO_MODIFICATION_ALLOWED_ERR` if `Node` is read only. • `HIERARCHY_REQUEST_ERR` if the `Node` is of a type that does not allow children of the type of the `newChild`, or the `Node` is an ancestor of this one.
`boolean hasChildNodes()`	return `true` if the `Node` has any children, otherwise `false`.
`Node cloneNode( boolean deep )`	return a duplicate of this `Node`. The duplicate has a `null` parent `Node`. Cloning an `Element` copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, because the text is contained in a child `Text` node. Cloning any other type of node simply returns a copy of this node. If specified `deep` is `true`, then the (sub)tree is also duplicated. If it is false, then only the `Node` itself is duplicated.

3.2.15 org.w3c.dom.NodeList

Description

The `org.w3c.dom.NodeList` interface provides an abstraction of an ordered collection of Node(s), without defining or constraining the implementation.

Methods

`int getLength()`	return the number of items in the list.
`Node item(index i)`	return the Node at the specified `index`, or `null` if the specified index is out of bounds.

3.2.16 org.w3c.dom.Notation

Description

The `org.w3c.dom.Notation` interface represents a notation as declared in a DTD. A notation either declares, by name, the format of an unparsed entity (see section 4.7 of the XML 1.0 specification), or is used for formal declaration of Processing Instruction targets (see section 2.6 of the XML 1.0 specification). The nodeName attribute inherited from `Node` is set to the declared name of the notation.

Methods

`String getPublicId()`	return the value of the public identifier or `null`.
`String getSystemId()`	return the value of the system identifier or `null`.

3.2.17 org.w3c.dom.ProcessingInstruction

Description

The `org.w3c.dom.ProcessingInstruction` interface represents a "processing instruction" used in XML as a way to keep processor-specific information in the text of the document.

Methods

`String getTarget()`	return the target of the PI.
`String getData()`	return the data associated with the PI target, or `null`.
`void setData( String data )`	set the data associated with the target PI. This method may raise a `DOMexception` with type `NO_MODIFICATION_ALLOWED_ERR` if the `Node` is read only.

3.2.18 org.w3c.dom.Text

Description

The `org.w3c.dom.Text` interface extends org.`w3c.dom.CharacterData`.

The `Text` interface represents the textual content (termed character data in XML) of an `Element` or `Attr`. If there is no markup inside an element's content, the text is contained in a single object implementing the Text interface that is the only child of the element. If there is markup, it is parsed into a list of elements and `Text` nodes that form the list of children of the element.

When a document is first made available via the DOM, there is only one `Text` node for each block of text. Users may create adjacent `Text` nodes that represent the contents of a given element without any intervening markup, but should be aware that there is no way to represent the separations between these nodes in XML or HTML, so they will not generally persist between DOM editing sessions. The `normalize()` method on `Element` merges any such adjacent `Text` objects into a single node for each block of text; this is recommended before employing operations that depend on a particular document structure, such as navigation with XPointers.

Methods

`Text splitText( int offset ) throws DOMException`	break this `Text` node into two `Text` nodes at the specified offset, keeping both in the tree as siblings. This node then only contains all the content up to the offset point. A new `Text` node, inserted as the next sibling of this node, contains all the content at and after the offset point. This returns the newly created `Text Node`. This method may raise the following `DOMException`(s): • `INDEX_SIZE_ERR` if the `offset` specified is either negative or greater than the number of characters in the data. • `NO_MODIFICATION_ALLOWED_ERR` if the `Node` is read only.

CHAPTER 4

Javax.xml.* packages

4.1 Overview

Although both SAX and DOM provide broad functionality, they are not complete. This is a significant issue, affecting the ability to author a truly portable application using only these APIs. Also, it is desirable to allow the underlying implementation of the parser mechanism to be pluggable.

This specification extends the SAX and DOM APIs to provide a completely portable a functional API.

4.2 Parser API Definition(s)

The Parser Factory APIs provide a parser implementation-independent programming interface to enable application(s) to parse XML content.

4.2.1 javax.xml.parsers.FactoryException

Description

The `javax.xml.parsers.FactoryException` is a public class that extends `java.lang.RuntimeException`. Instances are typically thrown by the parser and DOM factory implementations' subclasses to signal and encapsulate a variety of checked and runtime exceptions that may occur while manipulating artifacts from a plugged implementation.

Constructors

`FactoryException( String s )`	create a new `FactoryException` with the `String` specified as an error message.
`FactoryException( Exception e )`	create a new `FactoryException` with the `Exception` specified as the (encapsulated) causal exception.
`FactoryException( Exception e, String s )`	create a new `FactoryException` with the `Exception` specified as the (encapsulated) causal exception and the `String` specified as an error message.

Methods

`String getMessage()`	return the message (if any) associated with this exception. If no message was specified and an `Exception` is encapsulated, this has the effect of invoking `getMessage()` on the encapsulated `Exception`.
`Exception getException()`	return the actual exception (if any) that caused this exception to be raised.

4.2.2 javax.xml.parsers.SAXParserFactory

Description

A particular SAX `Parser` implementation is "plugged" into the platform via `SAXParserFactory` in one of two ways

- as a platform default.
- through external specification by a system property named "`javax.xml.parsers.SAXParserFactory`", obtained using `java.lang.System.getProperty()`.

This property (or platform default) names a class that is a concrete subclass of `javax.xml.parsers.SAXParserFactory`. This subclass shall implement a public no-args constructor used by the base abstract class to create an instance of the factory using the `newInstance()` method defined below.

The platform default is only used if no external implementation is available.

Once an application has obtained a reference to a `SAXParserFactory`, it can use this to configure and obtain parser instances.

Static Methods

SAXParserFactory newInstance()	obtain a new instance of a SAXParserFactory. Use the class named in the system property "javax.xml.parsers.SAXParserFactory", or the platform default if none is defined. This method throws javax.xml.parsers.FactoryException if the implementation is not available or cannot be instantiated.

Methods

void setNamespaceAware(boolean awareness)	specify if the parsers used by this SAXParserFactory are required to provide XML namespace support or not. This method throws IllegalArgumentException if the underlying implementation cannot provide the namespace conformance capability requested
void setLocale(Locale l)	set the SAXParserFactory Locale. The Locale may be used by the parser(s) implementing this SAXParserFactory in order to report any errors in a Locale-specific fashion. If a particular implementation cannot support the Locale specified, it may ignore this property.
void setValidating(boolean validating)	specify if the parsers used by this SAXParserFactory are required to validate the XML they parse. This method throws IllegalArgumentException if the underlying implementation cannot provide the validation capability requested.

boolean isNamespaceAware()	indicate if the SAXParserFactory is currently supporting XML Namespaces or not.
boolean isValidating()	indicate if the SAXParserFactory is using a validating XML parser or not.
Locale getLocale()	return the current Locale of the SAXParserFactory.

Abstract Methods

boolean checkValidating(boolean b)	check that the underlying implementation can support the validation capability specified.
boolean checkNamespaceAwareness(boolean b)	check that the underlying implementation can support the namespace conformance capability specified.
SAXParser newSAXParser() throws SAXException	create a new instance of SAXParser using the currently configured factory parameters. throws SAXException if the initialization of the underlying Parser fails.
org.xml.sax.Parser newParser	create a new instance of Parser using the currently configured factory parameters

These methods are implemented by concrete subclasses of this abstract base class.

4.2.3 javax.xml.parsers.SAXParser

Description

The javax.xml.parsers.SAXParser is a public class. It defines a convenience API that wraps an org.xml.sax.Parser that enables an application to parse XML content using, or to obtain, the actual parser instance wrapped.

This class implements a protected no-args constructor. Implementations are required to subclass this class in to provide their own implementation, returning instances of the same from the SAXParserFactory.newSAXParser() method.

Methods

void parse(InputStream is, HandlerBase hb) throws SAXException, IOException	parse the content of the `java.io.InputStream` instance as XML using the parser instance with the specified `org.xml.sax.HandlerBase` providing the implementations of `DocumentHandler`, `ErrorHandler`, `EntityResolver`, and `DTDHandler`. If any IO errors occur, an `IOException` shall be thrown. An `IllegalArgumentException` is thrown if the `InputStream` is `null`.
void parse(String uri, HandlerBase hb) throws SAXException, IOException	parse the content of the specified URI as XML using the parser instance with the specified `org.xml.sax.HandlerBase` providing the implementations of; `DocumentHandler`, `ErrorHandler`, `EntityResolver`, and `DTDHandler`. If any IO errors occur, an `IOException` shall be thrown. An `IllegalArgumentException` is thrown if the `URL` is `null`.
void parse(File f, HandlerBase hb) throws SAXException, IOException	parse the content of the`java.io.InputStream` instance as XML using the parser instance with the specified `org.xml.sax.HandlerBase` providing the implementations of `DocumentHandler`, `ErrorHandler`, `EntityResolver`, and `DTDHandler`. If any IO errors occur, an `IOException` shall be thrown. An `IllegalArgumentException` is thrown if the `File` is `null`.

void parse(InputSource is, HandlerBase hb) throws SAXException, IOException	parse the content of the `org.xml.sax.InputSource` instance as XML using the parser instance with the specified `org.xml.sax.HandlerBase` providing the implementations of `DocumentHandler`, `ErrorHandler`, `EntityResolver`, and `DTDHandler`. If any IO errors occur, an `IOException` shall be thrown. An `IllegalArgumentException` is thrown if the `InputStream` is `null`.
org.xml.sax.Parser parser()	return the actual `Parser` object wrapped by this instance.
boolean isNamespaceAware()	return if the `SAXParser` is supporting XML Namespaces or not.
boolean isValidating()	return if the `SAXParser` is using a validating XML parser or not.
Locale getLocale()	return the current `Locale` of the `SAXParser`.

4.2.4 javax.xml.parsers.DocumentBuilderFactory

Description

The `javax.xml.parsers.DocumentBuilderFactory` is an abstract public class. It provides a factory API that enables an application to obtain a `javax.xml.parsers.DocumentBuilder` object.

A particular Document Builder implementation is “plugged” into the platform in one of two ways:

- as a platform default.
- through external specification by a system property named “`javax.xml.parsers.DocumentBuilderFactory`” and obtained using `java.lang.System.getProperty()`.

This property (or platform default) names a subclass of `javax.xml.parsers.DocumentBuilderFactory`. This subclass shall implement a public no-args constructor used by this class to instantiate a factory using the `newInstance()` method defined below.

The platform default is only used if no external implementation is available.

Static Methods

`DocumentBuilderFactory newInstance()`	obtain a new instance of a `DocumentBuilderFactory`. Use the class named in the system property "`javax.xml.parsers.DocumentBuilderFactory`" or the platform default if none is defined. This method throws `javax.xml.FactoryException` if the implementation is not available or cannot be instantiated for any reason.

Methods

void setNamespaceAware(boolean awareness)	specify if the parser(s) used by this `DocumentBuilderFactory` shall be required to provide XML namespace support. If the value specified cannot be supported by the implementation, this method shall throw an `IllegalArgumentException`.
void setLocale(Locale l)	set the `DocumentBuilderFactory` `Locale`. The parser(s) implementing this `DocumentBuilderFactory` may use the `Locale` to report any errors in a Locale-specific fashion. If the value specified cannot be supported by the implementation, it may be silently ignored.
void setValidating(boolean validating)	specify if the parser(s) used by this `DocumentBuilderFactory` shall be required to validate the XML they parse. If the value specified cannot be supported by the implementation, an `IllegalArgumentException` shall be thrown.
boolean isNamespaceAware()	indicate if the `DocumentBuilderFactory` is currently supporting XML Namespaces or not.
boolean isValidating()	indicate if the `DocumentBuilderFactory` is using a validating XML parser or not.
Locale getLocale()	return the current `Locale` of the `DocumentBuilderFactory.`

Abstract Methods

boolean checkValidating(boolean b)	check that the underlying implementation can support the validation capability specified.
boolean checkNamespaceAwareness(boolean b)	check that the underlying implementation can support the namespace conformance capability specified.
boolean checkLocale(Locale l)	check that the underlying implementation can support the `Locale` specified.
DocumentBuilder newDocumentBuilder()	obtain a new instance of `DocumentBuilder` from the underlying implementation.

The methods above are implemented by concrete subclasses of this abstract base class.

4.2.5 javax.xml.parsers.DocumentBuilder

Description

The `javax.xml.parsers.DocumentBuilder` is an abstract public class. It provides a convenience API that enables an application to parse XML into, and obtain, `org.w3c.dom.Document` instances.

A `DocumentBuilder` instance is obtained from a `DocumentBuilderFactory` by invoking its `newDocumentBuilder()` method.

Implementations extend this base class to provide the parser- and document-dependent implementation(s).

Note that the `DocumentBuilder` reuses 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 `Document`. It merely requires that the implementation communicate with the application using these existing APIs.

Methods

void setEntityResolver(org.xml.SAX.EntityResolver er)	specify the `EntityResolver` to be used by this `DocumentBuilder`. Setting the `EntityResolver` to `null` will cause the underlying implementation to use its own default implementation and behavior. Changing this value during parsing does not affect the current operation.
void setErrorHandler(org.xml.SAX.ErrorHandler eh)	specify the `ErrorHandler` to be used by this `DocumentBuilder`. Setting the `ErrorHandler` to `null` will cause the underlying implementation to use its own default implementation and behavior. Changing this value during parsing does not affect the current operation.
org.w3c.dom.Document parse(InputStream is) throws SAXException, IOException	parse the content of the `java.io.InputStream` instance as XML using the associated parser instance and return a new `Document` containing a representation of the content parsed. If any parse errors or warnings occur, a `SAXException` shall be thrown. If any IO errors occur, an `IOException` shall be thrown. An `IllegalArgumentException` shall be thrown if the `InputStream` is `null`.

org.w3c.dom.Document parse(String uri) throws SAXException, IOException	parse the content at the URI specified as XML, using the associated parser instance, and return a new `Document` containing a representation of the content parsed. If any parse errors or warnings occur, then a `SAXException` shall be thrown. If any IO errors occur, then an `IOException` shall be thrown. An `IllegalArgumentException` shall be thrown if the `URL` is `null`.
org.w3c.dom.Document parse(File f) throws SAXException, IOException	parse the content of the `File` specified as XML, using the associated parser instance, and return a new `Document` containing a representation of the content parsed therein. If any parse errors or warnings occur, a `SAXException` shall be thrown. If any IO errors occur, an `IOException` shall be thrown. An `IllegalArgumentException` shall be thrown if the `File` is `null`.
org.w3c.dom.Document parse(InputSource is) throws SAXException, IOException	parse the content of the `org.xml.sax.InputSource` instance as XML using a distinct parser instance and return a `Document` representing the XML structure parsed. If any parse errors or warnings occur, a `SAXException` shall be thrown. If any IO errors occur, an `IOException` shall be thrown. An `IllegalArgumentException` shall be thrown if the `InputSource` is `null`.

`boolean isNamespaceAware()`	return if the `DocumentBuilder` is currently supporting XML Namespaces or not.
`boolean isValidating()`	return if the `DocumentBuilder` is using a validating XML parser or not.
`Locale getLocale()`	return the `Locale` of the `DocumentBuilder.`

Abstract Methods

`org.w3c.dom.Document parseDocument( InputSource is ) throws SAXException, IOException`	parse the content of the `org.xml.sax.InputSource` instance as XML using a distinct parser instance and return a `Document` representing the XML structure parsed. If any parse errors or warnings occur, a `SAXException` shall be thrown. If any IO errors occur, an `IOException` shall be thrown. An `IllegalArgumentException` shall be thrown if the `InputSource` is `null`.
`org.w3c.dom.Docunent newDocument()`	create a new `Document` instance.

These methods above are implemented by concrete subclasses of this abstract base class.

CHAPTER 5

XML & Namespace Conformance

5.1 Overview

This chapter describes the parser implementation well-formedness, validity, and namespaces conformance requirements.

Parser implementations that are accessed via the APIs defined here shall implement these constraints (without exception) to provide a predictable environment for application development and deployment.

5.2 Document Character Set Encoding(s)

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 PI: `<?XML version=1.0 encoding="<enc>" ?>`

The XML specification defines the following encoding values:

- `"UTF-8"`
- `"UTF-16"`
- `"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”
- “ASCII” (Note that ASCII encoded documents do not require an explicit encoding declaration in the XML declaration PI.)

Parser implementations are required to support the following encodings; ASCII, UTF-8 and UTF-16. Furthermore, parsers may optionally support additional encodings (including those defined above).

It is an error for a document to declare a particular encoding and actually use another.

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.

5.3 Parser Well Formedness Constraints

The W3C XML Specification (version 1.0) defines a “well formed” XML document to be a textual object that:

- Taken as a whole, matches the *document* production defined therein.
- Meets all the well-formedness constraints defined therein.
- References, either directly or indirectly, only parsed entities that are also well-formed..

Validating and non-validating parser implementations conforming to this standard specification are required to report any violations of the well-formedness constraints defined by the XML 1.0 specification.

5.4 Parser Validity Constraints

In addition to checking XML 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 associated DTD (if any)
- the XML validity constraints defined in the XML 1.0 Specification document.

5.5 Parser Namespace Support

XML namespaces are designed to be used to differentiate instances of markup within a single document.

Parser implementations may optionally[1] provide support to parse documents that utilize the W3C XML Namespaces Technical Recommendation. Conforming documents replace the XML syntactic production for *Name* with *QName*. XML elements and attributes may be comprised from a (possibly defaulted, and thus implicit) "*namespace prefix*," associated with a unique "*namespace URI*" defined by a "*namespace declaration*," and a "*local part*" separated by a single ":" character (when the namespace is other than the default). Entity names, processing instruction targets, and notation names shall not contain any ":" characters.

5.5.1 non-validating parser conformance

A non-validating parser that implements namespace support as defined is required to check for, and report as an error, any syntactic violation(s) defined by the W3C XML Namespace Specification. 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 an parsing error.

5.5.2 validating parser conformance

In addition to meeting the requirements for a non-validating parser, a validating parser that implements namespace support as defined is required to check for, and report as an error, any namespace used but not declared within a document, or its internal or external DTD (sub)set(s).

5.6 XML Namespace Prefix Usage

This standard extension reserves the XML namespace prefixes beginning with `java` and `javax` (case insensitive) for future usage by the Java(tm) Platform.

1. This may become mandatory in a future version of this API specification