

G

SAX 2.0.2 Reference

This appendix contains the specification of the SAX interface, version 2.0.2, some of which is explained in Chapter 12. It is taken largely verbatim from the definitive specification to be found at www.saxproject.org, with editorial comments added in italics. The classes and interfaces are described in alphabetical order and include the primary SAX interfaces and classes and SAX extensions. Deprecated classes and helper classes that are distributed with SAX 2.0.2 are not covered.

The SAX specification is in the public domain. (See the website mentioned previously for a statement of policy on copyright.) Essentially, the policy says do what you like with it, copy it as you wish, but no one accepts any liability for errors or omissions.

SAX 2.0.2 contains complete namespace support, which is available by default from any `XMLReader` object. An XML reader can also optionally supply raw XML 1.0 names. An XML reader is fully configurable: It is possible to attempt to query or change the current value of any feature or property. Features and properties are identified by fully qualified URIs, and parties are free to invent their own names for new extensions.

The `ContentHandler` and `Attributes` interfaces are similar to the deprecated `DocumentHandler` and `AttributeList` interfaces, but they add support for namespace-related information. `ContentHandler` also adds a callback for skipped entities, and the `Attributes` interface adds the capability to look up an attribute's index by name.

The following interfaces were included in SAX 1.0 but have been deprecated:

- `org.xml.sax.Parser`
- `org.xml.sax.DocumentHandler`
- `org.xml.sax.AttributeList`
- `org.xml.sax.HandlerBase`

These interfaces are not covered in this appendix, as their use is not widespread.

Classes and Interfaces

SAX is comprised of three packages. The first package, `org.xml.sax`, includes all the default interfaces and classes required to create a basic SAX application. The second package, `org.xml.sax.helpers`, provides default implementations of the interfaces as well as additional classes that can be used to support namespaces and work with legacy parsers. The third package, `org.xml.sax.ext`, includes extension interfaces that provide applications with additional details from the XML document. The following classes are included in this appendix.

From the `org.xml.sax` package:

- `Attributes`
- `ContentHandler`
- `DTDHandler`
- `EntityResolver`
- `ErrorHandler`
- `InputSource`
- `Locator`
- `SAXException`
- `SAXNotRecognizedException`
- `SAXNotSupportedException`
- `SAXParseException`
- `XMLFilter`
- `XMLReader`

From the `org.xml.sax.ext` package:

- `DeclHandler`
- `LexicalHandler`

The following extension interfaces were added in version 2.0.1:

- `Attributes2`
- `EntityResolver2`
- `Locator2`

More detailed explanation appears in Chapter 12 for the commonly used interfaces and classes.

Interface *org.xml.sax.Attributes*

This interface for a list of XML attributes enables access to a list of attributes in three different ways:

- ❑ By attribute index
- ❑ By namespace-qualified name
- ❑ By qualified (prefixed) name

The list will not contain attributes that were declared `#IMPLIED` but not specified in the start-tag. Nor will it contain attributes used as namespace declarations (`xmlns*`) unless the `http://xml.org/sax/features/namespace-prefixes` feature is set to `true` (it is `false` by default). Because SAX 2.0.2 conforms to the original “Namespaces in XML” recommendation, it normally does not give namespace declaration attributes a namespace URI.

Some SAX 2.0.2 parsers may support using an optional feature flag (`http://xml.org/sax/features/xmlns-uris`) to request that those attributes be given URIs, conforming to a later backwardly incompatible revision of that recommendation. (The attribute’s “local name” will be the prefix, or “xmlns” when defining a default element namespace.) For portability, handler code can be created that emulates or masks this feature, rather than require the use of parsers that can change the setting of that feature flag.

If the `namespace-prefixes` feature is `false`, then access by a qualified name may not be available; if the `http://xml.org/sax/features/namespaces` feature is `false`, then access by namespace-qualified names may not be available.

This interface replaces the now deprecated `SAX AttributeList` interface, which does not contain namespace support. In addition to namespace support, it adds the `getIndex` methods (covered in the following table).

The order of attributes in the list is unspecified and varies from implementation to implementation.

Method Name	Description
<code>getIndex(String)</code>	<pre>public int getIndex(String qName)</pre> Look up the index of an attribute by XML qualified name. <p>Parameters: <code>qName</code>: The qualified (prefixed) name.</p> <p>Returns: The index of the attribute, or -1 if it does not appear in the list.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>getIndex(String, String)</code>	<pre>public int getIndex(String uri, String localName)</pre> <p>Look up the index of an attribute by namespace name.</p> <p>Parameters: <code>uri</code>: The namespace URI, or the empty string if the name has no namespace URI.</p> <p><code>localName</code>: The attribute's local name.</p> <p>Returns: The index of the attribute, or -1 if it does not appear in the list.</p>
<code>getLength</code>	<pre>public int getLength()</pre> <p>Return the number of attributes in the list. Once you know the number of attributes, you can iterate through the list.</p> <p>Returns: The number of attributes in the list.</p>
<code>getLocalName(int)</code>	<pre>public String getLocalName(int index)</pre> <p>Look up an attribute's local name by index.</p> <p>Parameters: <code>index</code>: The attribute index (zero-based).</p> <p>Returns: The local name, or the empty string if namespace processing is not being performed, or null if the index is out of range.</p>
<code>getQName(int)</code>	<pre>public String getQName(int index)</pre> <p>Look up an attribute's XML qualified name by index.</p> <p>Parameters: <code>index</code>: The attribute index (zero-based).</p> <p>Returns: The XML qualified name, or the empty string if none is available, or null if the index is out of range.</p>

Method Name	Description
<code>GetType(int)</code>	<p><code>public String getType(int index)</code></p> <p>Look up an attribute's type by index.</p> <p>The attribute type is one of the strings "CDATA", "ID", "IDREF", "IDREFS", "NMOKEN", "NMTOKENS", "ENTITY", "ENTITIES", or "NOTATION" (always in uppercase).</p> <p>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 stated in the XML 1.0 Recommendation (clause 3.3.3, "Attribute-Value Normalization").</p> <p>For an enumerated attribute that is not a notation, the parser reports the type as "NMOKEN".</p> <p>Parameters: index: The attribute index (zero-based).</p> <p>Returns: The attribute's type as a string, or null if the index is out of range.</p>
<code>GetType(String)</code>	<p><code>public String getType(String qName)</code></p> <p>Look up an attribute's type by XML 1.0 qualified name.</p> <p>See <code>getType(int)</code> for a description of the possible types.</p> <p>Parameters: qName: The XML 1.0 qualified name.</p> <p>Returns: The attribute type as a string, or null if the attribute is not in the list or if qualified names are not available.</p>
<code>GetType(String, String)</code>	<p><code>public String getType(String uri, String localName)</code></p> <p>Look up an attribute's type by namespace name.</p> <p>See <code>getType(int)</code> for a description of the possible types.</p> <p>Parameters: uri: The namespace URI, or the empty string if the name has no namespace URI. localName: The attribute's local name.</p> <p>Returns: The attribute type as a string, or null if the attribute is not in the list or if namespace processing is not being performed.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>getURI(int)</code>	<pre>public String getURI(int index)</pre> <p>Look up an attribute's namespace URI by index.</p> <p>Parameters: <code>index</code>: The attribute index (zero-based).</p> <p>Returns: The namespace URI, or the empty string if none is available, or null if the index is out of range.</p>
<code>getValue(int)</code>	<pre>public String getValue(int index)</pre> <p>Look up an attribute's value by index.</p> <p>If the attribute value is a list of tokens (<code>IDREFS</code>, <code>ENTITIES</code>, or <code>NMTOKENS</code>), then the tokens are concatenated into a single string, with each token separated by a single space.</p> <p>Parameters: <code>index</code>: The attribute index (zero-based).</p> <p>Returns: The attribute's value as a string, or null if the index is out of range.</p>
<code>getValue(String)</code>	<pre>public String getValue(String qName)</pre> <p>Look up an attribute's value by XML 1.0 qualified name.</p> <p>See <code>getValue(int)</code> for a description of the possible values.</p> <p>Parameters: <code>qName</code>: The XML 1.0 qualified name.</p> <p>Returns: The attribute value as a string, or null if the attribute is not in the list or if qualified names are not available.</p>
<code>getValue(String, String)</code>	<pre>public String getValue(String uri, String localName)</pre> <p>Look up an attribute's value by namespace name.</p> <p>See <code>getValue(int)</code> for a description of the possible values.</p> <p>Parameters: <code>uri</code>: The namespace URI, or the empty string if the name has no namespace URI. <code>localName</code>: The attribute's local name.</p> <p>Returns: The attribute value as a string, or null if the attribute is not in the list.</p>

Interface *org.xml.sax.ext.Attributes2*

This is a SAX extension to augment the per-attribute information provided through attributes. If an implementation supports this extension, then the attributes provided in `ContentHandler.startElement()` implement this interface, and the `http://xml.org/sax/features/use-attributes2` feature flag will have the value `true`.

XMLReader implementations are not required to support this information, and it is not part of core-only SAX distributions.

Note that if an attribute was defaulted (`isSpecified()` is `false`), then it will of necessity also have been declared (`isDeclared()` is `true`) in the DTD. Similarly, if an attribute's type is anything except `CDATA`, then it must have been declared.

Method Name	Description
<code>isDeclared(int)</code>	<p><code>public boolean isDeclared(int index)</code></p> <p>Returns <code>false</code> unless the attribute was declared in the DTD. This helps distinguish two kinds of attributes that SAX reports as <code>CDATA</code>: ones that were declared (and hence are usually valid), and those that were not (and which are never valid).</p> <p>Parameters: Index: The attribute index (zero-based).</p> <p>Returns: <code>true</code> if the attribute was declared in the DTD, <code>false</code> otherwise.</p> <p>Throws: <code>ArrayIndexOutOfBoundsException</code>: When the supplied index does not identify an attribute.</p>
<code>isDeclared(String)</code>	<p><code>public boolean isDeclared(String qName)</code></p> <p>Returns <code>false</code> unless the attribute was declared in the DTD. This helps distinguish two kinds of attributes that SAX reports as <code>CDATA</code>: ones that were declared (and hence are usually valid), and those that were not (and which are never valid).</p> <p>Parameters: qName: The XML 1.0 qualified name.</p> <p>Returns: <code>true</code> if the attribute was declared in the DTD, <code>false</code> otherwise.</p> <p>Throws: <code>IllegalArgumentException</code>: When the supplied name does not identify an attribute.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>isDeclared</code> <code>(String, String)</code>	<pre>public boolean isDeclared(String uri, String localName)</pre> <p>Returns <code>false</code> unless the attribute was declared in the DTD. This helps distinguish two kinds of attributes that SAX reports as <code>CDATA</code>: ones that were declared (and hence are usually valid), and those that were not (and which are never valid).</p> <p>Remember that since DTDs do not “understand” namespaces, the namespace URI associated with an attribute may not have come from the DTD. The declaration will have applied to the attribute’s qualified name.</p> <p>Parameters: <code>uri</code>: The namespace URI, or the empty string if the name has no namespace URI. <code>localName</code>: The attribute’s local name.</p> <p>Returns: <code>true</code> if the attribute was declared in the DTD, <code>false</code> otherwise.</p> <p>Throws: <code>IllegalArgumentException</code>: When the supplied names do not identify an attribute.</p>
<code>isSpecified(int)</code>	<pre>public boolean isSpecified(int index)</pre> <p>Returns <code>true</code> unless the attribute value was provided by DTD defaulting.</p> <p>Parameters: <code>Index</code>: The attribute index (zero-based).</p> <p>Returns: <code>true</code> if the value was found in the XML text, <code>false</code> if the value was provided by DTD defaulting.</p> <p>Throws: <code>ArrayIndexOutOfBoundsException</code>: When the supplied index does not identify an attribute.</p>
<code>isSpecified</code> <code>(String)</code>	<pre>public boolean isSpecified(String qName)</pre> <p>Returns <code>true</code> unless the attribute value was provided by DTD defaulting.</p> <p>Parameters: <code>qName</code>: The XML 1.0 qualified name.</p> <p>Returns: <code>true</code> if the value was found in the XML text, <code>false</code> if the value was provided by DTD defaulting.</p>

Method Name	Description
	<p>Throws: <code>IllegalArgumentException</code>: When the supplied name does not identify an attribute.</p>
<code>isSpecified</code> <code>(String, String)</code>	<p><code>public boolean isSpecified(String uri, String localName)</code></p> <p>Returns <code>true</code> unless the attribute value was provided by DTD defaulting.</p> <p>Remember that since DTDs do not “understand” namespaces, the namespace URI associated with an attribute may not have come from the DTD. The declaration will have applied to the attribute’s qualified name.</p> <p>Parameters: <code>uri</code>: The namespace URI, or the empty string if the name has no namespace URI. <code>localName</code>: The attribute’s local name.</p> <p>Returns: <code>true</code> if the value was found in the XML text, <code>false</code> if the value was provided by DTD defaulting.</p> <p>Throws: <code>IllegalArgumentException</code>: When the supplied names do not identify an attribute.</p>

Interface `org.xml.sax.ContentHandler`

This interface enables you to receive notification of the logical content of a document.

This is the main interface that most SAX applications implement: If the application needs to be informed of basic parsing events, it implements this interface and registers an instance with the SAX parser using the `setContentHandler` method. The parser uses the instance to report basic document-related events such as the start and end of elements and character data.

The order of events in this interface is important, and mirrors the order of information in the document itself. For example, all of an element’s content (character data, processing instructions, and/or sub-elements) will appear, in order, between the `startElement` event and the corresponding `endElement` event.

This interface is similar to the now deprecated SAX 1.0 `DocumentHandler` interface, but it adds support for namespaces and for reporting skipped entities (in nonvalidating XML processors).

Implementors should note that there is also a Java class `ContentHandler` in the `java.net` package; therefore, it’s probably a bad idea to do the following:

```
import java.net.*;
import org.xml.sax.*;
```

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>characters</code> (<code>char[]</code> , <code>int</code> , <code>int</code>)	<pre>public void characters(char[] ch, int start, int length)</pre> <p>throws <code>SAXException</code></p> <p>Receive notification of character data.</p> <p>The parser calls this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all the characters in any single event must come from the same external entity so that the <code>Locator</code> provides useful information.</p> <p>The application must not attempt to read from the array outside of the specified range.</p> <p>Individual characters may consist of more than one Java <code>char</code> value. There are two important cases where this happens, because characters can't be represented in just sixteen bits. In one case, characters are represented in a <i>surrogate pair</i>, using two special Unicode values. Such characters are in the so-called "Astral Planes," with a code point above <code>U+FFFF</code>. A second case involves composite characters, such as a base character combining with one or more accent characters.</p> <p>Your code should not assume that algorithms using <code>char</code>-at-a-time idioms will be working in character units; in some cases they split characters. This is relevant wherever XML permits arbitrary characters, such as attribute values, processing instruction data, and comments, as well as in data reported from this method. It's also generally relevant whenever Java code manipulates internationalized text; the issue isn't unique to XML.</p> <p>Note that some parsers report whitespace in element content using the <code>ignorableWhitespace</code> method, rather than this one (validating parsers must do so).</p> <p>Parameters: <code>ch</code>: The characters from the XML document. <code>start</code>: The start position in the array. <code>length</code>: The number of characters to read from the array.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception.</p>

Method Name	Description
<code>endDocument</code>	<p><code>public void endDocument ()</code></p> <p>throws <code>SAXException</code></p> <p>Receive notification of the end of a document.</p> <p>There is an apparent contradiction between the documentation for this method and the documentation for <code>ErrorHandler.fatalError</code>. Until this ambiguity is resolved in a future major release, clients should make no assumptions about whether <code>endDocument</code> will or will not be invoked when the parser has reported a <code>fatalError</code> or thrown an exception.</p> <p>The SAX parser invokes this method only once, and it will be the last method invoked during the parse. If the parser does call this method, then it will not invoke it until it has either abandoned parsing (because of an unrecoverable error) or reached the end of input.</p> <p>For more information, see the discussion in Chapter 12.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception.</p>
<code>endElement (String, String, String)</code>	<p><code>public void endElement (String uri, String localName, String qName)</code></p> <p>throws <code>SAXException</code></p> <p>Receive notification of the end of an element.</p> <p>The SAX parser invokes this method at the end of every element in the XML document; there will be a corresponding <code>startElement</code> event for every <code>endElement</code> event (even when the element is empty).</p> <p>For information on the names, see <code>startElement</code>.</p> <p>Parameters: <code>uri</code>: The namespace URI, or the empty string if the element has no namespace URI or if namespace processing is not being performed. <code>localName</code>: The local name (without prefix), or the empty string if namespace processing is not being performed. <code>qName</code>: The qualified XML 1.0 name (with prefix), or the empty string if qualified names are not available.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>endPrefixMapping(String)</code>	<p><code>public void endPrefixMapping(String prefix)</code></p> <p>throws <code>SAXException</code></p> <p>End the scope of a prefix-URI mapping.</p> <p>See <code>startPrefixMapping</code> for details. These events always occur immediately after the corresponding <code>endElement</code> event, but the order of <code>endPrefixMapping</code> events is not otherwise guaranteed.</p> <p>Parameters: <code>prefix</code>: The prefix that was being mapped. This is the empty string when a default mapping scope ends.</p> <p>Throws: <code>SAXException</code>: The client may throw an exception during processing.</p>
<code>ignorableWhitespace(char[], int, int)</code>	<p><code>public void ignorableWhitespace(char[] ch, int start, int length)</code></p> <p>throws <code>SAXException</code></p> <p>Receive notification of ignorable whitespace in element content.</p> <p>Validating parsers must use this method to report each chunk of whitespace in element content (see the W3C XML 1.0 Recommendation, section 2.10); nonvalidating parsers may also use this method if they are capable of parsing and using content models.</p> <p>SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the locator provides useful information.</p> <p>The application must not attempt to read from the array outside of the specified range.</p> <p>Parameters: <code>ch</code>: The characters from the XML document. <code>start</code>: The start position in the array. <code>length</code>: The number of characters to read from the array.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception.</p>

Method Name	Description
<code>processingInstruction(String, String)</code>	<pre>public void processingInstruction(String target, String data)</pre> <p>throws <code>SAXException</code></p> <p>Receive notification of a processing instruction.</p> <p>The parser will invoke this method once for each processing instruction found. Note that processing instructions may occur before or after the main document element.</p> <p>A SAX parser must never report an XML declaration (XML 1.0, section 2.8) or a text declaration (XML 1.0, section 4.3.1) using this method.</p> <p>Like <code>characters()</code>, processing instruction data may have characters that need more than one char value.</p> <p>Parameters: <code>target</code>: The processing instruction target. <code>data</code>: The processing instruction data, or null if none was supplied. The data does not include any whitespace separating it from the target.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception.</p>
<code>setDocumentLocator(Locator)</code>	<pre>public void setDocumentLocator(Locator locator)</pre> <p>Receive an object for locating the origin of SAX document events.</p> <p>SAX parsers are strongly encouraged (though not absolutely required) to supply a locator: if it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the <code>ContentHandler</code> interface.</p> <p>The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application uses this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.</p> <p>Note that the locator returns correct information only during the invocation SAX event callbacks after <code>startDocument</code> returns and before <code>endDocument</code> is called. The application should not attempt to use it at any other time.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>setDocumentLocator(Locator)</code>	Parameters: <code>locator</code> : An object that can return the location of any SAX document event.
<code>skippedEntity(String)</code>	<pre>public void skippedEntity(String name)</pre> <p>throws <code>SAXException</code></p> <p>Receive notification of a skipped entity. This is not called for entity references within markup constructs, such as element start-tags or markup declarations. (The XML Recommendation requires reporting skipped external entities. SAX also reports internal entity expansion/nonexpansion, except within markup constructs.)</p> <p>The parser invokes this method each time the entity is skipped. Nonvalidating processors may skip entities if they have not seen the declarations (because, for example, the entity was declared in an external DTD subset). All processors may skip external entities, depending on the values of the <code>http://xml.org/sax/features/external-general-entities</code> and the <code>http://xml.org/sax/features/external-parameter-entities</code> properties.</p> Parameters: <code>name</code> : The name of the skipped entity. If it is a parameter entity, then the name begins with '%', and if it is the external DTD subset, then it is the string "[dtd]". Throws: <code>SAXException</code> : Any SAX exception, possibly wrapping another exception.
<code>startDocument()</code>	<pre>public void startDocument()</pre> <p>throws <code>SAXException</code></p> <p>Receive notification of the beginning of a document.</p> <p>The SAX parser will invoke this method only once, before any other event callbacks (except for <code>setDocumentLocator</code>).</p> Throws: <code>SAXException</code> : Any SAX exception, possibly wrapping another exception.

Method Name	Description
<code>startElement (String, String, String, Attributes)</code>	<p> <code>public void startElement(String uri, String localName, String qName, Attributes atts)</code> </p> <p>throws <code>SAXException</code></p> <p>Receive notification of the beginning of an element.</p> <p>The parser will invoke this method at the beginning of every element in the XML document; there will be a corresponding <code>endElement</code> event for every <code>startElement</code> event (even when the element is empty). All of the element's content will be reported, in order, before the corresponding <code>endElement</code> event.</p> <p>This event allows up to three name components for each element:</p> <ul style="list-style-type: none"> The namespace URI The local name The qualified (prefixed) name <p>Any or all of these may be provided, depending on the values of the <code>http://xml.org/sax/features/namespaces</code> and <code>http://xml.org/sax/features/namespace-prefixes</code> properties:</p> <p>The namespace URI and local name are required when the <code>namespaces</code> feature is <code>true</code> (the default), and are optional when the <code>namespaces</code> feature is <code>false</code> (if one is specified, then both must be);</p> <p>The qualified name is required when the <code>namespace-prefixes</code> feature is <code>true</code>, and is optional when the <code>namespace-prefixes</code> feature is <code>false</code> (the default).</p> <p>Note that the attribute list provided will contain only attributes with explicit values (specified or defaulted); <code>#IMPLIED</code> attributes will be omitted. The attribute list will contain attributes used for namespace declarations (<code>xmlns*</code> attributes) only if the <code>http://xml.org/sax/features/namespace-prefixes</code> feature is <code>true</code> (it is <code>false</code> by default, and support for a <code>true</code> value is optional).</p> <p>Like <code>characters()</code>, attribute values may have characters that need more than one char value.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>startElement</code> (String, String, String, Attributes)	<p>Parameters:</p> <p><code>uri</code>: The namespace URI, or the empty string if the element has no namespace URI or if namespace processing is not being performed.</p> <p><code>localName</code>: The local name (without prefix), or the empty string if namespace processing is not being performed.</p> <p><code>qName</code>: The qualified XML 1.0 name (with prefix), or the empty string if qualified names are not available.</p> <p><code>atts</code>: The attributes attached to the element. If there are no attributes, it shall be an empty <code>Attributes</code> object. The value of this object after <code>startElement</code> returns is undefined.</p> <p>Throws:</p> <p><code>SAXException</code>: Any SAX exception, possibly wrapping another exception.</p>
<code>startPrefix</code> <code>Mapping</code> (String, String)	<pre>public void startPrefixMapping(String prefix, String uri) throws SAXException</pre> <p>Begin the scope of a prefix-URI namespace mapping.</p> <p>The information from this event is not necessary for normal namespace processing: The SAX <code>XMLReader</code> will automatically replace prefixes for element and attribute names when the <code>http://xml.org/sax/features/namespaces</code> feature is <code>true</code> (the default).</p> <p>There are cases, however, when applications need to use prefixes in character data or attribute values, where they cannot safely be expanded automatically; the <code>start/endPrefixMapping</code> event supplies the information to the application to expand prefixes in those contexts itself, if necessary.</p> <p>Note that <code>start/endPrefixMapping</code> events are not guaranteed to be properly nested relative to each other: all <code>startPrefixMapping</code> events occur immediately before the corresponding <code>startElement</code> event, and all <code>endPrefixMapping</code> events occur immediately after the corresponding <code>endElement</code> event, but their order is not otherwise guaranteed.</p> <p>There should never be <code>start/endPrefixMapping</code> events for the <code>xml</code> prefix, since it is predeclared and immutable.</p> <p>Parameters:</p> <p><code>prefix</code>: The namespace prefix being declared. An empty string is used for the default element namespace, which has no prefix.</p> <p><code>uri</code>: The namespace URI the prefix is mapped to.</p> <p>Throws:</p> <p><code>SAXException</code>: The client may throw an exception during processing.</p>

Interface *org.xml.sax.ext.DeclHandler*

This interface is the SAX extension handler for DTD declaration events.

It is an optional extension handler for SAX to provide more complete information about DTD declarations in an XML document. XML readers are not required to recognize this handler, and it is not part of core-only SAX distributions.

Note that data-related DTD declarations (unparsed entities and notations) are already reported through the *DTDHandler* interface. If you are using the declaration handler with a lexical handler, all the events will occur between the *startDTD* and the *endDTD* events.

To set the *DeclHandler* for an XML reader, use the *setProperty* method with the property name `http://xml.org/sax/properties/declaration-handler` and an object implementing this interface (or null) as the value. If the reader does not report declaration events, it will throw a *SAXNotRecognizedException* when you attempt to register the handler.

Method Name	Description
<code>attributeDecl (String, String, String, String, String)</code>	<p><code>public void attributeDecl (String eName, String aName, String type, String mode, String value)</code></p> <p>throws <i>SAXException</i></p> <p>Report an attribute type declaration.</p> <p>Only the effective (first) declaration for an attribute will be reported. The type will be one of the strings <i>CDATA</i>, <i>ID</i>, <i>IDREF</i>, <i>IDREFS</i>, <i>NMTOKEN</i>, <i>NMTOKENS</i>, <i>ENTITY</i>, <i>ENTITIES</i>, a parenthesized token group with the separator <code> </code> and all whitespace removed, or the word <i>NOTATION</i> followed by a space followed by a parenthesized token group with all whitespace removed.</p> <p>The value will be the value as reported to applications, appropriately normalized and with entity and character references expanded.</p> <p>Parameters:</p> <p><code>eName</code>: The name of the associated element.</p> <p><code>aName</code>: The name of the attribute.</p> <p><code>type</code>: A string representing the attribute type.</p> <p><code>mode</code>: A string representing the attribute defaulting mode (<i>#IMPLIED</i>, <i>#REQUIRED</i>, or <i>#FIXED</i>) or null if none of these applies.</p> <p><code>value</code>: A string representing the attribute's default value, or null if there is none.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>attributeDecl</code> (String, String, String, String, String)	Throws: SAXException: The application may raise an exception.
<code>elementDecl</code> (String, String)	<pre>public void elementDecl (String name, String model)</pre> <p>throws SAXException</p> <p>Report an element type declaration.</p> <p>The content model consists of the string <code>EMPTY</code>, the string <code>ANY</code>, or a parenthesized group, optionally followed by an occurrence indicator. The model will be normalized so that all parameter entities are fully resolved and all whitespace is removed, and will include the enclosing parentheses. Other normalization (such as removing redundant parentheses or simplifying occurrence indicators) is at the discretion of the parser.</p> Parameters: name: The element type name. model: The content model as a normalized string. Throws: SAXException: The application may raise an exception.
<code>externalEntity</code> <code>Decl</code> (String, String, String)	<pre>public void externalEntityDecl (String name, String publicId, String systemId)</pre> <p>throws SAXException</p> <p>Report a parsed external entity declaration.</p> <p>Only the effective (first) declaration for each entity will be reported. If the system identifier is a URL, the parser must resolve it fully before passing it to the application.</p> Parameters: name: The name of the entity. If it is a parameter entity, the name will begin with <code>'%'</code> . publicId: The declared public identifier of the entity, or null if none was declared. systemId: The declared system identifier of the entity. Throws: SAXException: The application may raise an exception.

Method Name	Description
<code>internalEntityDecl(String, String)</code>	<p><code>public void internalEntityDecl (String name, String value)</code></p> <p>throws <code>SAXException</code></p> <p>Report an internal entity declaration.</p> <p>Only the effective (first) declaration for each entity will be reported. All parameter entities in the value will be expanded, but general entities will not.</p> <p>Parameters: name: The name of the entity. If it is a parameter entity, the name will begin with '%'. value: The replacement text of the entity.</p> <p>Throws: <code>SAXException</code>: The application may raise an exception.</p>

Interface *org.xml.sax.DTDHandler*

You use this interface to receive notification of basic DTD-related events.

If a SAX application needs information about notations and unparsed entities, then the application implements this interface and registers an instance with the SAX parser using the parser's `setDTDHandler` method. The parser uses the instance to report notation and unparsed entity declarations to the application.

Note that this interface includes only those DTD events that the XML recommendation requires processors to report: notation and unparsed entity declarations.

The SAX parser may report these events in any order, regardless of the order in which the notations and unparsed entities were declared; however, all DTD events must be reported after the document handler's `startDocument` event, and before the first `startElement` event. (If the `LexicalHandler` is used, these events must also be reported before the `endDTD` event.)

It is up to the application to store the information for future use (perhaps in a hash table or object tree). If the application encounters attributes of type `NOTATION`, `ENTITY`, or `ENTITIES`, it can use the information that it obtained through this interface to find the entity and/or notation corresponding with the attribute value.

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>notationDecl (String, String, String)</code>	<pre>public void notationDecl (String name, String publicId, String systemId) throws SAXException</pre> <p>Receive notification of a notation declaration event.</p> <p>It is up to the application to record the notation for later reference, if necessary; notations may appear as attribute values and in unparsed entity declarations, and are sometime used with processing instruction target names.</p> <p>At least one of <code>publicId</code> and <code>systemId</code> must be non-null. If a system identifier is present, and it is a URL, then the SAX parser must resolve it fully before passing it to the application through this event.</p> <p>There is no guarantee that the notation declaration will be reported before any unparsed entities that use it.</p> <p>Parameters: <code>name</code>: The notation name. <code>publicId</code>: The notation's public identifier, or null if none was given. <code>systemId</code>: The notation's system identifier, or null if none was given.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception.</p>
<code>unparsedEntity Decl (String, String, String, String)</code>	<pre>public void unparsedEntityDecl (String name, String publicId, String systemId, String notationName) throws SAXException</pre> <p>Receive notification of an unparsed entity declaration event.</p> <p>Note that the notation name corresponds to a notation reported by the <code>notationDecl</code> event. It is up to the application to record the entity for later reference, if necessary; unparsed entities may appear as attribute values.</p> <p>If the system identifier is a URL, then the parser must resolve it fully before passing it to the application.</p>

Method Name	Description
<p>unparsedEntityDecl(String, String, String, String)</p>	<p>Parameters: name: The unparsed entity's name. publicId: The entity's public identifier, or null if none was given. systemId: The entity's system identifier. notationName: The name of the associated notation.</p> <p>Throws: SAXException: Any SAX exception, possibly wrapping another exception.</p>

Interface *org.xml.sax.EntityResolver*

This is a basic interface for resolving entities.

If a SAX application needs to implement customized handling for external entities, it must implement this interface and register an instance with the SAX driver using the `setEntityResolver` method. The XML reader will then allow the application to intercept any external entities (including the external DTD subset and external parameter entities, if any) before including them.

Many SAX applications will not need to implement this interface, but it is especially useful for applications that build XML documents from databases or other specialized input sources, or for applications that use URI types other than URLs.

The following resolver would provide the application with a special character stream for the entity with the system identifier `http://www.myhost.com/today`:

```
import org.xml.sax.EntityResolver;
import org.xml.sax.InputSource;

public class MyResolver implements EntityResolver {
    public InputSource resolveEntity (String publicId, String systemId)
    {
        if (systemId.equals("http://www.myhost.com/today")) {
            // return a special input source
            MyReader reader = new MyReader();
            return new InputSource(reader);
        } else {
            // use the default behaviour
            return null;
        }
    }
}
```

The application can also use this interface to redirect system identifiers to local URIs or to look up replacements in a catalog (possibly by using the public identifier).

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>resolveEntity</code> (String, String)	<p><code>public InputSource resolveEntity(String publicId, String systemId)</code></p> <p>throws <code>SAXException</code>, <code>IOException</code></p> <p>Allow the application to resolve external entities.</p> <p>The parser will call this method before opening any external entity except the top-level document entity. Such entities include the external DTD subset and external parameter entities referenced within the DTD (in either case, only if the parser reads external parameter entities), and external general entities referenced within the document element (if the parser reads external general entities). The application may request that the parser locate the entity itself, that it use an alternative URI, or that it use data provided by the application (as a character or byte input stream).</p> <p>Application writers can use this method to redirect external system identifiers to secure and/or local URIs, to look up public identifiers in a catalogue, or to read an entity from a database or other input source (including, for example, a dialog box). Neither XML nor SAX specifies a preferred policy for using public or system IDs to resolve resources. However, SAX specifies how to interpret any <code>InputSource</code> returned by this method, and that if none is returned, the system ID will be dereferenced as a URL.</p> <p>If the system identifier is a URL, then the SAX parser must resolve it fully before reporting it to the application.</p> <p>Parameters: <code>publicId</code>: The public identifier of the external entity being referenced, or null if none was supplied. <code>systemId</code>: The system identifier of the external entity being referenced.</p> <p>Returns: An <code>InputSource</code> object describing the new input source, or null to request that the parser open a regular URI connection to the system identifier.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception. <code>IOException</code>: A Java-specific IO exception, possibly the result of creating a new <code>InputStream</code> or <code>Reader</code> for the <code>InputSource</code>, or an illegal URL.</p>

Interface `org.xml.sax.ext.EntityResolver2`

This is an extended interface for mapping external entity references to input sources, or providing a missing external subset. The `XMLReader.setEntityResolver()` method is used to provide implementations of this interface to parsers. When a parser uses the methods in this interface, the `EntityResolver2.resolveEntity()` method (in this interface) is used instead of the older `EntityResolver.resolveEntity()` method.

If a SAX application requires the customized handling, which this interface defines for external entities, it must ensure that it uses an `XMLReader` with the `http://xml.org/sax/features/use-entity-resolver2` feature flag set to `true` (which is its default value when the feature is recognized). If that flag is unrecognized, or its value is `false`, or the resolver does not implement this interface, then only the `EntityResolver` method will be used.

Method Name	Description
<code>getExternalSubset (String, String)</code>	<pre>public InputSource getExternalSubset (String name, String baseURI)</pre> <p>throws <code>SAXException</code>, <code>IOException</code></p> <p>Allows applications to provide an external subset for documents that don't explicitly define one. Documents with <code>DOCTYPE</code> declarations that omit an external subset can thus augment the declarations available for validation, entity processing, and attribute processing (normalization, defaulting, and reporting types including ID). This augmentation is reported through the <code>startDTD()</code> method as if the document text had originally included the external subset; this callback is made before any internal subset data or errors are reported.</p> <p>This method can also be used with documents that have no <code>DOCTYPE</code> declaration. When the root element is encountered but no <code>DOCTYPE</code> declaration has been seen, this method is invoked. If it returns a value for the external subset, that root element is declared to be the root element, with the effect of splicing a <code>DOCTYPE</code> declaration at the end of the prolog of a document that could not otherwise be valid. The sequence of parser callbacks in that case logically resembles this:</p> <pre>... comments and PIs from the prolog (as usual) startDTD ("rootName", source.getPublicId (), source.get- SystemId ()); startEntity (" [dtd] "); ... declarations, comments, and PIs from the external subset</pre>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>getExternalSubset (String, String)</code>	<pre>endEntity (" [dtd] "); endDTD (); ... then the rest of the document (as usual) startElement (... , "rootName", ...);</pre> <p>Note that the <code>InputSource</code> gets no further resolution. Implementations of this method may wish to invoke <code>resolveEntity()</code> to gain benefits such as use of local caches of DTD entities. Also, this method is never used by a (nonvalidating) processor that is not including external parameter entities.</p> <p>Uses for this method include facilitating data validation when interop-erating with XML processors that would always require undesirable network accesses for external entities, or which for other reasons adopt a “no DTDs” policy. Nonvalidation motives include forcing documents to include DTDs so that attributes are handled consistently. For exam-ple, an XPath processor needs to know which attributes have type "ID" before it can process a widely used type of reference.</p> <p>Warning: Returning an external subset modifies the input document. Providing definitions for general entities can make a malformed docu-ment appear to be well formed.</p> <p>Parameters: <code>name</code>: Identifies the document root element. This name comes from a DOCTYPE declaration (where available) or the actual root element. <code>baseURI</code>: The document’s base URI, serving as an additional hint for selecting the external subset. This is always an absolute URI, unless it is null because the <code>XMLReader</code> was given an <code>InputSource</code> without one.</p> <p>Returns: An <code>InputSource</code> object describing the new external subset to be used by the parser, or null to indicate that no external subset is provided.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception. <code>IOException</code>: A Java-specific IO exception, possibly the result of creating a new <code>InputStream</code> or <code>Reader</code> for the <code>InputSource</code>, or an illegal URL.</p>

Method Name	Description
<pre>resolveEntity (String, String, String)</pre>	<pre>public InputSource resolveEntity(String name, String publicId, String baseURI, String systemId)</pre> <p>throws SAXException, IOException</p> <p>Allows applications to map references to external entities into input sources, or to tell the parser it should use conventional URI resolution. This method is only called for external entities that have been properly declared. It provides more flexibility than the <code>EntityResolver</code> interface, supporting implementations of more complex catalogue schemes such as the one defined by the OASIS XML Catalogs specification.</p> <p>Parsers configured to use this resolver method will call it to determine the input source to use for any external entity being included because of a reference in the XML text. That excludes the document entity, and any external entity returned by <code>getExternalSubset()</code>. When a (non-validating) processor is configured not to include a class of entities (parameter or general) through use of feature flags, this method is not invoked for such entities.</p> <p>Note that the entity naming scheme used here is the same one used in the <code>LexicalHandler</code>, or in the <code>ContentHandler.skippedEntity()</code> method.</p> <p>Parameters: <code>name</code>: Identifies the external entity being resolved. Either <code>[dtd]</code> for the external subset, or a name starting with <code>%</code> to indicate a parameter entity, or else the name of a general entity. This is never null when invoked by a SAX parser. <code>publicId</code>: The public identifier of the external entity being referenced (normalized as required by the XML specification), or null if none was supplied. <code>baseURI</code>: The URI with respect to which relative system IDs are interpreted. This is always an absolute URI, unless it is null (likely because the <code>XMLReader</code> was given an <code>InputSource</code> without one). This URI is defined by the XML specification to be the one associated with the <code><</code> starting the relevant declaration. <code>systemId</code>: The system identifier of the external entity being referenced: either a relative or absolute URI. This is never null when invoked by a SAX parser; only declared entities, and any external subset, are resolved by such parsers.</p> <p>Returns: An <code>InputSource</code> object describing the new input source to be used by the parser. Returning null directs the parser to resolve the system ID against the base URI and open a connection to resulting URI.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>resolveEntity</code> (String, String, String)	Throws: SAXException: Any SAX exception, possibly wrapping another exception. IOException: A Java-specific IO exception, possibly the result of creating a new <code>InputStream</code> or <code>Reader</code> for the <code>InputSource</code> , or an illegal URL.

Interface `org.xml.sax.ErrorHandler`

This is a basic interface for SAX error handlers.

If a SAX application needs to implement customized error handling, then it must implement this interface and then register an instance with the `XMLReader` using the `setErrorHandler` method. The parser then reports all errors and warnings through this interface.

For XML processing errors, a SAX driver must use this interface in preference to throwing an exception: It is up to the application to decide whether to throw an exception for different types of errors and warnings. Note, however, that there is no requirement that the parser continue to report additional errors after a call to `fatalError`. In other words, a SAX driver class may throw an exception after reporting any `fatalError`. Also, parsers may throw appropriate exceptions for non-XML errors. For example, `XMLReader.parse()` would throw an `IOException` for errors accessing entities or the document.

Warning: *If an application does not register an `ErrorHandler`, XML parsing errors go unreported, except that `SAXParseException` is thrown for fatal errors. In order to detect validity errors, an `ErrorHandler` that does something with `error()` calls must be registered.*

Method Name	Description
<code>error(SAXParseException)</code>	<code>public void error(SAXParseException exception)</code> throws <code>SAXException</code> Receive notification of a recoverable error. This corresponds to the definition of “error” in Section 1.2 of the W3C XML 1.0 Recommendation. For example, a validating parser would use this callback to report the violation of a validity constraint. The default behavior is to take no action. Additionally, parsers that support XML 1.1 may report an error when a Unicode Normalization error is encountered. The SAX parser must continue to provide normal parsing events after invoking this method: It should still be possible for the application to process the document through to the end. If the application cannot do so, then the parser should report a fatal error even if the XML 1.0 recommendation does not require it to do so. Filters may use this method to report other, non-XML errors as well.

Method Name	Description
	<p>Parameters: <code>exception</code>: The error information encapsulated in a SAX parse exception.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception.</p>
<code>fatalError</code> (<code>SAXParseException</code>)	<p><code>public void fatalError (SAXParseException exception)</code></p> <p>throws <code>SAXException</code></p> <p>Receive notification of a nonrecoverable error.</p> <p>This corresponds to the definition of “fatal error” in Section 1.2 of the W3C XML 1.0 Recommendation. For example, a parser would use this callback to report the violation of a well-formedness constraint.</p> <p>The application must assume that the document is unusable after the parser has invoked this method, and should continue (if at all) only for the sake of collecting additional error messages: In fact, SAX parsers are free to stop reporting any other events once this method has been invoked.</p> <p>Parameters: <code>exception</code>: The error information encapsulated in a SAX parse exception.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception.</p>
<code>warning</code> (<code>SAXParseException</code>)	<p><code>public void warning (SAXParseException exception)</code></p> <p>throws <code>SAXException</code></p> <p>Receive notification of a warning.</p> <p>SAX parsers use this method to report conditions that are not errors or fatal errors as defined by the XML 1.0 Recommendation. The default behavior is to take no action.</p> <p>The SAX parser must continue to provide normal parsing events after invoking this method: It should still be possible for the application to process the document through to the end.</p> <p>Filters may use this method to report other, non-XML warnings as well.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>warning</code> (<code>SAXParseException</code>)	Parameters: <code>exception</code> : The warning information encapsulated in a SAX parse exception. Throws: <code>SAXException</code> : Any SAX exception, possibly wrapping another exception.

Class *org.xml.sax.InputSource*

This class is a single input source for an XML entity. It enables a SAX application to encapsulate information about an input source in a single object, which may include a public identifier, a system identifier, a byte stream (possibly with a specified encoding), and/or a character stream.

The application can deliver an input source to the parser as the return value of the `EntityResolver.resolveEntity` method.

The SAX parser uses the `InputSource` object to determine how to read XML input. If a character stream is available, the parser reads that stream directly, disregarding any text encoding declaration found in that stream. If there is no character stream but there is a byte stream, then the parser uses that byte stream, using the encoding specified in the `InputSource` or (if no encoding is specified) auto-detecting the character encoding using an algorithm such as the one in the XML Specification. If neither a character stream nor a byte stream is available, the parser will attempt to open a URI connection to the resource identified by the system identifier.

An `InputSource` object belongs to the application: The SAX parser shall never modify it in any way (it may modify a copy if necessary). However, standard processing of both byte and character streams is to close them as part of end-of-parse cleanup, so applications should not attempt to reuse such streams after they have been handed to a parser.

Constructor	Description
<code>InputSource</code>	<code>public InputSource()</code> Zero-argument default constructor.
<code>InputSource</code> (<code>InputStream</code>)	<code>public InputSource(InputStream byteStream)</code> Create a new input source with a byte stream. Application writers should use <code>setSystemId()</code> to provide a base for resolving relative URIs, may use <code>setPublicId</code> to include a public identifier, and may use <code>setEncoding</code> to specify the object's character encoding. Parameters: <code>byteStream</code> : The raw byte stream containing the document.

Constructor	Description
<code>InputSource (Reader)</code>	<pre>public InputSource(Reader characterStream)</pre> <p>Create a new input source with a character stream.</p> <p>Application writers should use <code>setSystemId()</code> to provide a base for resolving relative URIs, and may use <code>setPublicId</code> to include a public identifier.</p> <p>The character stream shall not include a byte order mark.</p> <p>Parameters: <code>characterStream</code>: The character stream containing the document.</p>
<code>InputSource (String)</code>	<pre>public InputSource(String systemId)</pre> <p>Create a new input source with a system identifier.</p> <p>Applications may use <code>setPublicId</code> to include a public identifier as well, or <code>setEncoding</code> to specify the character encoding, if known.</p> <p>If the system identifier is a URL, it must be fully resolved (it may not be a relative URL).</p> <p>Parameters: <code>systemId</code>: The system identifier (URI).</p>

Method Name	Description
<code>getByteStream</code>	<pre>public InputStream getByteStream()</pre> <p>Get the byte stream for this input source.</p> <p>The <code>getEncoding</code> method will return the character encoding for this byte stream, or null if unknown.</p> <p>Returns: The byte stream, or null if none was supplied.</p>
<code>getCharacter Stream</code>	<pre>public Reader getCharacterStream()</pre> <p>Get the character stream for this input source.</p> <p>Returns: The character stream, or null if none was supplied.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>getEncoding</code>	<pre>public String getEncoding ()</pre> <p>Get the character encoding for a byte stream or URI. This value is ignored when the application provides a character stream.</p> <p>Returns: The encoding, or null if none was supplied.</p>
<code>getPublicId</code>	<pre>public String getPublicId ()</pre> <p>Get the public identifier for this input source.</p> <p>Returns: The public identifier, or null if none was supplied.</p>
<code>getSystemId</code>	<pre>public String getSystemId ()</pre> <p>Get the system identifier for this input source.</p> <p>The <code>getEncoding</code> method will return the character encoding of the object pointed to, or null if unknown.</p> <p>If the system ID is a URL, then it will be fully resolved.</p> <p>Returns: The system identifier, or null if none was supplied.</p>
<code>setByteStream (InputStream)</code>	<pre>public void setByteStream (InputStream byteStream)</pre> <p>Set the byte stream for this input source.</p> <p>The SAX parser ignores this if there is also a character stream specified, but it will use a byte stream in preference to opening a URI connection itself.</p> <p>If the application knows the character encoding of the byte stream, it should set it with the <code>setEncoding</code> method.</p> <p>Parameters: <code>byteStream</code>: A byte stream containing an XML document or other entity.</p>
<code>setCharacter Stream(Reader)</code>	<pre>public void setCharacterStream (Reader characterStream)</pre> <p>Set the character stream for this input source.</p> <p>If a character stream is specified, then the SAX parser will ignore any byte stream and not attempt to open a URI connection to the system identifier.</p>

Method Name	Description
	<p>Parameters: <code>characterStream</code>: The character stream containing the XML document or other entity.</p>
<code>setEncoding</code> <code>(String)</code>	<p><code>public void setEncoding (String encoding)</code></p> <p>Set the character encoding, if known.</p> <p>The encoding must be a string acceptable for an XML encoding declaration (see Section 4.3.3 of the XML 1.0 Recommendation).</p> <p>This method has no effect when the application provides a character stream.</p> <p>Parameters: <code>encoding</code>: A string describing the character encoding.</p>
<code>setPublicId</code> <code>(String)</code>	<p><code>public void setPublicId (String publicId)</code></p> <p>Set the public identifier for this input source.</p> <p>The public identifier is always optional: If the application writer includes one, it will be provided as part of the location information.</p> <p>Parameters: <code>publicId</code>: The public identifier as a string.</p>
<code>setSystemId</code> <code>(String)</code>	<p><code>public void setSystemId (String systemId)</code></p> <p>Set the system identifier for this input source.</p> <p>The system identifier is optional if there is a byte stream or a character stream, but it is still useful to provide one, as the application can use it to resolve relative URIs and can include it in error messages and warnings (the parser will attempt to open a connection to the URI only when no byte stream or character stream is specified).</p> <p>If the application knows the character encoding of the object pointed to by the system identifier, then it can register the encoding using the <code>setEncoding</code> method.</p> <p>If the system identifier is a URL, then it must be fully resolved (it may not be a relative URL).</p> <p>Note: Though this is a SAX requirement, most implementations support relative URLs in the XML document.</p> <p>Parameters: <code>systemId</code>: The system identifier as a string.</p>

Interface *org.xml.sax.ext.LexicalHandler*

This is an optional SAX extension handler for SAX to provide lexical information about an XML document, such as comments and CDATA section boundaries. XML readers are not required to recognize this handler, and it is not part of core-only SAX distributions.

The events in the lexical handler apply to the entire document, not just to the document element, and all lexical handler events must appear between the content handler's `startDocument` and `endDocument` events.

To set the `LexicalHandler` for an `XMLReader`, use the `setProperty` method with the property name `http://xml.org/sax/properties/lexical-handler` and an object implementing this interface (or null) as the value. If the reader does not report lexical events, then it will throw a `SAXNotRecognizedException` when you attempt to register the handler.

Method Name	Description
<code>comment(char[], int, int)</code>	<pre>public void comment(char[] ch, int start, int length)</pre> <p>throws <code>SAXException</code></p> <p>Report an XML comment anywhere in the document.</p> <p>This callback is used for comments inside or outside the document element, including comments in the external DTD subset (if read). Comments in the DTD must be properly nested inside <code>start/endDTD</code> and <code>start/endEntity</code> events (if used).</p> <p>Parameters: <code>ch</code>: An array holding the characters in the comment. <code>start</code>: The start position in the array. <code>length</code>: The number of characters to read from the array.</p> <p>Throws: <code>SAXException</code>: The application may raise an exception.</p>
<code>endCDATA</code>	<pre>public void endCDATA()</pre> <p>throws <code>SAXException</code></p> <p>Report the end of a CDATA section.</p> <p>Throws: <code>SAXException</code>: The application may raise an exception.</p>

Method Name	Description
endDTD	<p><code>public void endDTD()</code></p> <p>throws <code>SAXException</code></p> <p>Report the end of DTD declarations.</p> <p>This method is intended to report the end of the DOCTYPE declaration; if the document has no DOCTYPE declaration, then this method is not invoked.</p> <p>Throws: <code>SAXException</code>: The application may raise an exception.</p>
endEntity(String)	<p><code>public void endEntity(String name)</code></p> <p>throws <code>SAXException</code></p> <p>Report the end of an entity.</p> <p>Parameters: name: The name of the entity that is ending.</p> <p>Throws: <code>SAXException</code>: The application may raise an exception.</p>
startCDATA	<p><code>public void startCDATA()</code></p> <p>throws <code>SAXException</code></p> <p>Report the start of a CDATA section.</p> <p>The contents of the CDATA section will be reported through the regular characters event; this event is intended only to report the boundary.</p> <p>Throws: <code>SAXException</code>: The application may raise an exception.</p>
startDTD(String, String, String)	<p><code>public void startDTD(String name, String publicId, String systemId)</code></p> <p>throws <code>SAXException</code></p> <p>Report the start of DTD declarations, if any.</p> <p>This method is intended to report the beginning of the DOCTYPE declaration; if the document has no DOCTYPE declaration, this method is not invoked.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>startDTD(String, String, String)</code>	<p>All declarations reported through <code>DTDHandler</code> or <code>DeclHandler</code> events must appear between the <code>startDTD</code> and <code>endDTD</code> events. Declarations are assumed to belong to the internal DTD subset unless they appear between <code>startEntity</code> and <code>endEntity</code> events. Comments and processing instructions from the DTD should also be reported between the <code>startDTD</code> and <code>endDTD</code> events, in their original order of (logical) occurrence; they are not required to appear in their correct locations relative to <code>DTDHandler</code> or <code>DeclHandler</code> events, however.</p> <p>Note that the <code>start/endDTD</code> events will appear within the <code>start/endDocument</code> events from <code>ContentHandler</code> and before.</p> <p>Parameters: <code>name</code>: The document type name. <code>publicId</code>: The declared public identifier for the external DTD subset, or null if none was declared. <code>systemId</code>: The declared system identifier for the external DTD subset, or null if none was declared. (Note that this is not resolved against the document base URI.)</p> <p>Throws: <code>SAXException</code>: The application may raise an exception.</p>
<code>startEntity(String)</code>	<pre>public void startEntity(String name) throws SAXException</pre> <p>Report the beginning of some internal and external XML entities.</p> <p>The reporting of parameter entities (including the external DTD subset) is optional, and SAX drivers that report <code>LexicalHandler</code> events may not implement it; you can use the http://xml.org/sax/features/lexical-handler/parameter-entities feature to query or control the reporting of parameter entities.</p> <p>General entities are reported with their regular names, parameter entities have % prepended to their names, and the external DTD subset has the pseudo-entity name [dtd].</p> <p>When a SAX driver is providing these events, all other events must be properly nested within <code>start/endEntity</code> events. There is no additional requirement that events from <code>DeclHandler</code> or <code>DTDHandler</code> be properly ordered.</p>

Method Name	Description
	Note that skipped entities are reported through the <code>skippedEntity</code> event, which is part of the <code>ContentHandler</code> interface.
	<p>Because of the streaming event model that SAX uses, the following types of entity boundaries cannot be reported under any circumstances:</p> <ul style="list-style-type: none"> General entities within attribute values Parameter entities within declarations <p>These will be silently expanded, with no indication of where the original entity boundaries were. Note also that the boundaries of character references are not reported.</p> <p>All <code>start/endEntity</code> events must be properly nested.</p> <p>Parameters: <code>name</code>: The name of the entity. If it is a parameter entity, then the name begins with <code>%</code>, and if it is the external DTD subset, then it is <code>[dtd]</code>.</p> <p>Throws: <code>SAXException</code>: The application may raise an exception.</p>

Interface *org.xml.sax.Locator*

This is an interface for associating a SAX event with a document location.

If a SAX parser provides location information to the SAX application, then it does so by implementing this interface and then passing an instance to the application using the content handler's `setDocumentLocator` method. The application can use the object to obtain the location of any other SAX event in the XML source document.

Note that the results returned by the object are valid only during the scope of each callback method: The application will receive unpredictable results if it attempts to use the locator at any other time or after parsing completes.

SAX parsers are not required to supply a locator, but they are very strongly encouraged to do so. If the parser supplies a locator, then it must do so before reporting any other document events. If no locator has been set by the time the application receives the `startDocument` event, then the application should assume that a locator is not available.

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>getColumnNumber</code>	<pre>public int getColumnNumber()</pre> <p>Return the column number where the current document event ends. This is a one-based number of Java char values since the last line end.</p> <p>Warning: The return value from the method is intended only as an approximation for the sake of diagnostics; it is not intended to provide sufficient information to edit the character content of the original XML document. For example, when lines contain combining character sequences, wide characters, surrogate pairs, or bi-directional text, the value may not correspond to the column in a text editor's display.</p> <p>The return value is an approximation of the column number in the document entity or external parsed entity where the markup triggering the event appears.</p> <p>If possible, the SAX driver should provide the line position of the first character after the text associated with the document event. The first column in each line is column 1.</p> <p>Returns: The column number, or -1 if none is available.</p>
<code>getLineNumber</code>	<pre>public int getLineNumber()</pre> <p>Return the line number where the current document event ends. Lines are delimited by line ends, which are defined in the XML Specification.</p> <p>Warning: The return value from the method is intended only as an approximation for the sake of diagnostics; it is not intended to provide sufficient information to edit the character content of the original XML document. In some cases, these "line" numbers match what would be displayed as columns, and in others they may not match the source text due to internal entity expansion.</p> <p>The return value is an approximation of the line number in the document entity or external parsed entity where the markup triggering the event appears.</p> <p>If possible, the SAX driver should provide the line position of the first character after the text associated with the document event. The first line is line 1.</p> <p>Returns: The line number, or -1 if none is available.</p>

Method Name	Description
<code>getPublicId</code>	<pre>public String getPublicId()</pre> <p>Return the public identifier for the current document event.</p> <p>The return value is the public identifier of the document entity or the external parsed entity in which the markup triggering the event appears.</p> <p>Returns: A string containing the public identifier, or null if none is available.</p>
<code>getSystemId</code>	<pre>public String getSystemId()</pre> <p>Return the system identifier for the current document event.</p> <p>The return value is the system identifier of the document entity or the external parsed entity in which the markup triggering the event appears.</p> <p>If the system identifier is a URL, then the parser must resolve it fully before passing it to the application. For example, a filename must always be provided as a <code>file://</code> URL, and other kinds of relative URI are also resolved against their bases.</p> <p>Returns: A string containing the system identifier, or null if none is available.</p>

Interface `org.xml.sax.ext.Locator2`

This is a SAX extension to augment the entity information provided through a `Locator`. If an implementation supports this extension, then the `Locator` provided in `ContentHandler.setDocumentLocator()` will implement this interface, and the `http://xml.org/sax/features/use-locator2` feature flag will have the value `true`.

`XMLReader` implementations are not required to support this information, and it is not part of core-only SAX distributions.

Method Name	Description
<code>getEncoding</code>	<pre>public String getEncoding()</pre> <p>Returns the name of the character encoding for the entity. If the encoding was declared externally (for example, in a MIME Content-Type header), then that will be the name returned. If there was an <code><?xml ... encoding='...' ?></code> declaration at the start of the document, then that encoding name will be returned. Otherwise, the encoding will be inferred (normally to be UTF-8, or some UTF-16 variant), and that inferred name will be returned.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>getEncoding</code>	<p>When an <code>InputSource</code> is used to provide an entity's character stream, this method returns the encoding provided in that input stream.</p> <p>Note that some recent W3C specifications require that text in some encodings be normalized, using Unicode Normalization Form C, before processing. Such normalization must be performed by applications, and would normally be triggered based on the value returned by this method.</p> <p>Encoding names may be those used by the underlying virtual machine, and comparisons should be case insensitive.</p> <p>Returns: Name of the character encoding used to interpret the entity's text, or null if this was not provided for a character stream passed through an <code>InputSource</code> or if this was otherwise not yet available in the current parsing state.</p>
<code>getXMLVersion</code>	<p><code>public String getXMLVersion()</code></p> <p>Returns the version of XML used for the entity. This will normally be the identifier from the current entity's <code><?xml version='...' ...?></code> declaration, or is defaulted by the parser.</p> <p>Returns: Identifier for the XML version being used to interpret the entity's text, or null if that information is not yet available in the current parsing state.</p>

***Exception* org.xml.sax.SAXException**

This class encapsulates a general SAX error or warning. It can contain basic error or warning information from the XML parser or the application: A parser writer or application writer can subclass it to provide additional functionality. SAX handlers may throw this exception or any exception subclassed from it.

If the application needs to pass through other types of exceptions, then it must wrap those exceptions in a `SAXException` or an exception derived from a `SAXException`. If the parser or application needs to include information about a specific location in an XML document, it should use the `SAXParseException` subclass.

Constructor	Description
<code>SAXException</code>	<p><code>public SAXException()</code></p> <p>Create a new <code>SAXException</code>.</p>

Constructor	Description
SAXException (Exception)	<pre>public SAXException(Exception e)</pre> <p>Create a new SAXException wrapping an existing exception.</p> <p>The existing exception will be embedded in the new one, and its message becomes the default message for the SAXException.</p> <p>Parameters: e: The exception to be wrapped in a SAXException.</p>
SAXException (String)	<pre>public SAXException(String message)</pre> <p>Create a new SAXException.</p> <p>Parameters: message: The error or warning message.</p>
SAXException (String, Exception)	<pre>public SAXException(String message, Exception e)</pre> <p>Create a new SAXException from an existing exception.</p> <p>The existing exception will be embedded in the new one, but the new exception will have its own message.</p> <p>Parameters: message: The detail message. e: The exception to be wrapped in a SAXException.</p>

Method Name	Description
getException	<pre>public Exception getException()</pre> <p>Return the embedded exception, if any.</p> <p>Returns: The embedded exception, or null if there is none.</p>
getMessage	<pre>public String getMessage()</pre> <p>Return a detail message for this exception.</p> <p>If there is an embedded exception and if the SAXException has no detail message of its own, then this method returns the detail message from the embedded exception.</p> <p>Returns: The error or warning message.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>toString</code>	<code>public String toString()</code> Override <code>toString</code> to pick up any embedded exception. Returns: A string representation of this exception.

Exception `org.xml.sax.SAXNotRecognizedException`

This is an exception class for an unrecognized identifier.

An `XMLReader` will throw this exception when it finds an unrecognized feature or property identifier; SAX applications and extensions may use this class for other similar purposes.

Constructor	Description
<code>SAXNotRecognizedException</code>	<code>public SAXNotRecognizedException()</code> Construct a new exception with no message.
<code>SAXNotRecognizedException(String)</code>	<code>public SAXNotRecognizedException(String message)</code> Construct a new exception with the given message. Parameters: message: The text message of the exception.

Exception `org.xml.sax.SAXNotSupportedException`

This is the exception class for an unsupported operation.

An `XMLReader` will throw this exception when it recognizes a feature or property identifier, but cannot perform the requested operation (setting a state or value). Other SAX applications and extensions may use this class for similar purposes.

Constructor	Description
<code>SAXNotSupportedException</code>	<code>public SAXNotSupportedException()</code> Construct a new exception with no message.
<code>SAXNotSupportedException(String)</code>	<code>public SAXNotSupportedException(String message)</code> Construct a new exception with the given message. Parameters: message: The text message of the exception.

Exception *org.xml.sax.SAXParseException*

This exception encapsulates an XML parse error or warning. It may include information for locating the error in the original XML document, as if it came from a `Locator` object. Note that although the application will receive a `SAXParseException` as the argument to the handlers in the `ErrorHandler` interface, the application is not actually required to throw the exception; instead, it can simply read the information in it and take a different action.

Since this exception is a subclass of `SAXException`, it inherits the ability to wrap another exception.

Constructor	Description
<code>SAXParseException</code> (<code>String</code> , <code>Locator</code>)	<pre>public SAXParseException(String message, Locator locator)</pre> <p>Create a new <code>SAXParseException</code> from a message and a <code>Locator</code>.</p> <p>This constructor is especially useful when an application is creating its own exception from within a <code>ContentHandler</code> callback.</p> <p>Parameters: message: The error or warning message. locator: The <code>locator</code> object for the error or warning (may be null).</p>
<code>SAXParseException</code> (<code>String</code> , <code>Locator</code> , <code>Exception</code>)	<pre>public SAXParseException(String message, Locator locator, Exception e)</pre> <p>Wrap an existing exception in a <code>SAXParseException</code>.</p> <p>This constructor is especially useful when an application is creating its own exception from within a <code>ContentHandler</code> callback and needs to wrap an existing exception that is not a subclass of <code>SAXException</code>.</p> <p>Parameters: message: The error or warning message, or null to use the message from the embedded exception. locator: The <code>locator</code> object for the error or warning (may be null). e: Any exception.</p>
<code>SAXParseException</code> (<code>String</code> , <code>String</code> , <code>String</code> , <code>int</code> , <code>int</code>)	<pre>public SAXParseException(String message, String publicId, String systemId, int lineNumber, int columnNumber)</pre> <p>Create a new <code>SAXParseException</code>.</p> <p>This constructor is most useful for parser writers.</p> <p>All parameters except the message are as if they were provided by a <code>Locator</code>. For example, if the system identifier is a URL (including relative filename), then the caller must resolve it fully before creating the exception.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Constructor	Description
<code>SAXParseException</code> (String, String, String, int, int)	Parameters: message: The error or warning message. publicId: The public identifier of the entity that generated the error or warning. systemId: The system identifier of the entity that generated the error or warning. lineNumber: The line number of the end of the text that caused the error or warning. columnNumber: The column number of the end of the text that caused the error or warning.
<code>SAXParseException</code> (String, String, String, int, int, Exception)	<code>public SAXParseException(String message, String publicId, String systemId, int lineNumber, int columnNumber, Exception e)</code> Creates a new <code>SAXParseException</code> with an embedded exception. This constructor is most useful for parser writers who need to wrap an exception that is not a subclass of <code>SAXException</code> . All parameters except the message and exception are as if they were provided by a <code>Locator</code> . For example, if the system identifier is a URL (including relative filename), then the caller must resolve it fully before creating the exception. Parameters: message: The error or warning message, or null to use the message from the embedded exception. publicId: The public identifier of the entity that generated the error or warning. systemId: The system identifier of the entity that generated the error or warning. lineNumber: The line number of the end of the text that caused the error or warning. columnNumber: The column number of the end of the text that caused the error or warning. e: Another exception to embed in this one.

Method Name	Description
<code>getColumnNumber</code>	<pre>public int getColumnNumber()</pre> <p>The column number of the end of the text where the exception occurred.</p> <p>The first column in a line is position 1.</p> <p>Returns: An integer representing the column number, or -1 if none is available.</p>
<code>getLineNumber</code>	<pre>public int getLineNumber()</pre> <p>The line number of the end of the text where the exception occurred.</p> <p>The first line is line 1.</p> <p>Returns: An integer representing the line number, or -1 if none is available.</p>
<code>getPublicId</code>	<pre>public String getPublicId()</pre> <p>Get the public identifier of the entity where the exception occurred.</p> <p>Returns: A string containing the public identifier, or null if none is available.</p>
<code>getSystemId</code>	<pre>public String getSystemId()</pre> <p>Get the system identifier of the entity where the exception occurred.</p> <p>If the system identifier is a URL, then it will have been resolved fully.</p> <p>Returns: A string containing the system identifier, or null if none is available.</p>

Interface org.xml.sax.XMLFilter

This is an interface for an XML filter.

An XMLFilter is like an XMLReader, except that it obtains its events from another XMLReader rather than from a primary source such as an XML document or database. Filters can modify a stream of events as they pass on to the final application.

The XMLFilterImpl helper class provides a convenient base for creating SAX filters, by passing on all EntityResolver, DTDHandler, ContentHandler, and ErrorHandler events automatically.

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>getParent</code>	<pre>public XMLReader getParent ()</pre> <p>Get the parent reader.</p> <p>This method enables the application to query the parent reader (which may be another filter). It is generally a bad idea to perform any operations on the parent reader directly: They should all pass through this filter.</p> <p>Returns: The parent filter, or null if none has been set.</p>
<code>setParent (XMLReader)</code>	<pre>public void setParent (XMLReader parent)</pre> <p>Set the parent reader.</p> <p>This method enables the application to link the filter to a parent reader (which may be another filter). The argument may not be null.</p> <p>Parameters: <code>reader</code> - The parent reader.</p>

Interface *org.xml.sax.XMLReader*

This is an interface for reading an XML document using callbacks.

Note: Despite its name, this interface does not extend the standard Java `Reader` interface, because reading XML is a fundamentally different activity than reading character data.

`XMLReader` is the interface that an XML parser's SAX driver must implement. This interface enables an application to set and query features and properties in the parser, to register event handlers for document processing, and to initiate a document parse.

All SAX interfaces are assumed to be synchronous: The parse methods must not return until parsing is complete, and readers must wait for an event-handler callback to return before reporting the next event.

This interface replaces the (now deprecated) SAX 1.0 `Parser` interface. The `XMLReader` interface contains two important enhancements over the old `Parser` interface (as well as some minor ones):

- ❑ It adds a standard way to query and set features and properties.
- ❑ It adds namespace support, which is required for many higher-level XML standards.

Method Name	Description
<code>getContentHandler</code>	<p><code>public ContentHandler getContentHandler ()</code></p> <p>Return the current content handler.</p> <p>Returns: The current content handler, or null if none has been registered.</p>
<code>getDTDHandler</code>	<p><code>public DTDHandler getDTDHandler ()</code></p> <p>Return the current DTD handler.</p> <p>Returns: The current DTD handler, or null if none has been registered.</p>
<code>getEntityResolver</code>	<p><code>public EntityResolver getEntityResolver ()</code></p> <p>Return the current entity resolver.</p> <p>Returns: The current entity resolver, or null if none has been registered.</p>
<code>getErrorHandler</code>	<p><code>public ErrorHandler getErrorHandler ()</code></p> <p>Return the current error handler.</p> <p>Returns: The current error handler, or null if none has been registered.</p>
<code>getFeature (String)</code>	<p><code>public boolean getFeature (String name)</code></p> <p>throws <code>SAXNotRecognizedException</code>,</p> <p><code>SAXNotSupportedException</code></p> <p>Look up the value of a feature flag.</p> <p>The feature name is any fully qualified URI. It is possible for an <code>XMLReader</code> to recognize a feature name but temporarily be unable to return its value. Some feature values may be available only in specific contexts, such as before, during, or after a parse. Also, some feature values may not be programmatically accessible. (In the case of an adapter for SAX 1.0 <code>Parser</code>, there is no implementation-independent way to expose whether the underlying parser is performing validation, expanding external entities, and so forth.)</p> <p>All <code>XMLReaders</code> are required to recognize the <code>http://xml.org/sax/features/namespaces</code> and <code>http://xml.org/sax/features/namespace-prefixes</code> feature names.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>getFeature(String)</code>	<p>Typical usage is something like this:</p> <pre>XMLReader r = new MySAXDriver(); // try to activate validation try { r.setFeature("http://xml.org/sax/features/validation", true); } catch (SAXException e) { System.err.println("Cannot activate feature."); } // register event handlers r.setContentHandler(new MyContentHandler()); r.setErrorHandler(new MyErrorHandler()); // parse the first document try { r.parse("http://www.foo.com/mydoc.xml"); } catch (IOException e) { System.err.println("I/O exception reading XML"); } catch (SAXException e) { System.err.println("XML error in document."); } </pre> <p>Implementers are free (and encouraged) to invent their own features, using names built on their own URIs.</p>

Method Name	Description
	<p>Parameters: name: The feature name, which is a fully qualified URI.</p> <p>Returns: The current value of the feature (true or false).</p> <p>Throws: SAXNotRecognizedException: If the feature value can't be assigned or retrieved. SAXNotSupportedException: When the XMLReader recognizes the feature name but cannot determine its value at this time.</p>
<pre>getProperty (String)</pre>	<pre>public Object getProperty(String name) throws SAXNotRecognizedException, SAXNotSupportedException</pre> <p>Look up the value of a property.</p> <p>The property name is any fully qualified URI. It is possible for an XMLReader to recognize a property name but temporarily be unable to return its value. Some property values may be available only in specific contexts, such as before, during, or after a parse.</p> <p>XMLReaders are not required to recognize any specific property names, though an initial core set is documented for SAX.</p> <p>Implementers are free (and encouraged) to invent their own properties, using names built on their own URIs.</p> <p>Parameters: name: The property name, which is a fully qualified URI.</p> <p>Returns: The current value of the property.</p> <p>Throws: SAXNotRecognizedException: If the property value can't be assigned or retrieved. SAXNotSupportedException: When the XMLReader recognizes the property name but cannot determine its value at this time.</p>
<pre>parse(InputSource)</pre>	<pre>public void parse(InputSource input) throws IOException, SAXException</pre> <p>Parse an XML document.</p>

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>parse (InputSource)</code>	<p>The application can use this method to instruct the XML reader to begin parsing an XML document from any valid input source (a character stream, a byte stream, or a URI).</p> <p>Applications may not invoke this method while a parse is in progress (they should create a new <code>XMLReader</code> instead for each nested XML document). Once a parse is complete, an application may reuse the same <code>XMLReader</code> object, possibly with a different input source. Configuration of the <code>XMLReader</code> object (such as handler bindings and values established for feature flags and properties) is unchanged by completion of a parse, unless the definition of that aspect of the configuration explicitly specifies other behavior (e.g., feature flags or properties exposing characteristics of the document being parsed).</p> <p>During the parse, the <code>XMLReader</code> provides information about the XML document through the registered event handlers.</p> <p>This method is synchronous: It won't return until parsing has ended. If a client application wants to terminate parsing early, then it should throw an exception.</p> <p>Parameters: <code>input</code>: The input source for the top level of the XML document.</p> <p>Throws: <code>SAXException</code>: Any SAX exception, possibly wrapping another exception. <code>IOException</code>: An IO exception from the parser, possibly from a byte stream or character stream supplied by the application.</p>
<code>parse (String)</code>	<pre>public void parse (String systemId)</pre> <p>throws <code>IOException</code>, <code>SAXException</code></p> <p>Parse an XML document from a system identifier (URI).</p> <p>This method is a shortcut for the common case of reading a document from a system identifier. It is the exact equivalent of the following:</p> <pre>parse (new InputSource (systemId)) ;</pre> <p>If the system identifier is a URL, then it must be fully resolved by the application before it is passed to the parser.</p> <p>Parameters: <code>systemId</code>: The system identifier (URI).</p>

Method Name	Description
	<p>Throws: SAXException: Any SAX exception, possibly wrapping another exception. IOException: An IO exception from the parser, possibly from a byte stream or character stream supplied by the application.</p>
setContentHandler (ContentHandler)	public void setContentHandler (ContentHandler handler) Allow an application to register a content event handler. If the application doesn't register a content handler, then all content events reported by the SAX parser are silently ignored. Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately. Parameters: handler: The content handler.
setDTDHandler (DTDHandler)	public void setDTDHandler (DTDHandler handler) Allow an application to register a DTD event handler. If the application does not register a DTD handler, then all DTD events reported by the SAX parser are silently ignored. Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately. Parameters: handler: The DTD handler.
setEntityResolver (EntityResolver)	public void setEntityResolver (EntityResolver resolver) Allow an application to register an entity resolver. If the application does not register an entity resolver, then the XML-Reader will perform its own default resolution. Applications may register a new or different resolver in the middle of a parse, and the SAX parser must begin using the new resolver immediately. Parameters: resolver: The entity resolver.

Table continued on following page

Appendix G: SAX 2.0.2 Reference

Method Name	Description
<code>setErrorHandler</code> (<code>ErrorHandler</code>)	<pre>public void setErrorHandler (ErrorHandler handler)</pre> <p>Allow an application to register an error event handler.</p> <p>If the application does not register an error handler, then all error events reported by the SAX parser are silently ignored; however, normal processing may not continue. It is highly recommended that all SAX applications implement an error handler to avoid unexpected bugs.</p> <p>Applications may register a new or different handler in the middle of a parse, and the SAX parser must begin using the new handler immediately.</p> <p>Parameters: handler: The error handler.</p>
<code>setFeature</code> (<code>String</code> , <code>boolean</code>)	<pre>public void setFeature (String name, boolean value)</pre> <p>throws <code>SAXNotRecognizedException</code>,</p> <p><code>SAXNotSupportedException</code></p> <p>Set the value of a feature flag.</p> <p>The feature name is any fully qualified URI. It is possible for an <code>XMLReader</code> to expose a feature value but to be unable to change the current value. Some feature values may be immutable or mutable only in specific contexts, such as before, during, or after a parse.</p> <p>All <code>XMLReaders</code> are required to support setting <code>http://xml.org/sax/features/namespaces</code> to <code>true</code> and <code>http://xml.org/sax/features/namespaces-prefixes</code> to <code>false</code>.</p> <p>Parameters: name: The feature name, which is a fully qualified URI. value: The requested value of the feature (<code>true</code> or <code>false</code>).</p> <p>Throws: <code>SAXNotRecognizedException</code>: If the feature value can't be assigned or retrieved. <code>SAXNotSupportedException</code>: When the <code>XMLReader</code> recognizes the feature name but cannot set the requested value.</p>

Method Name	Description
<code>setProperty (String, Object)</code>	<p data-bbox="554 231 1200 254"><code>public void setProperty(String name, Object value)</code></p> <p data-bbox="554 289 1003 312">throws <code>SAXNotRecognizedException</code>,</p> <p data-bbox="582 347 918 370"><code>SAXNotSupportedException</code></p> <p data-bbox="554 403 841 426">Set the value of a property.</p> <p data-bbox="554 462 1315 575">The property name is any fully qualified URI. It is possible for an <code>XMLReader</code> to recognize a property name but to be unable to change the current value. Some property values may be immutable or mutable only in specific contexts, such as before, during, or after a parse.</p> <p data-bbox="554 610 1315 663"><code>XMLReaders</code> are not required to recognize setting any specific property names, though a core set is defined by SAX.</p> <p data-bbox="554 698 1253 751">This method is also the standard mechanism for setting extended handlers.</p> <p data-bbox="554 786 686 809">Parameters:</p> <p data-bbox="554 813 1162 836">name: The property name, which is a fully qualified URI.</p> <p data-bbox="554 839 1033 862">value: The requested value for the property.</p> <p data-bbox="554 897 648 920">Throws:</p> <p data-bbox="554 924 1322 977"><code>SAXNotRecognizedException</code>: If the property value can't be assigned or retrieved.</p> <p data-bbox="554 1012 1300 1065"><code>SAXNotSupportedException</code>: When the <code>XMLReader</code> recognizes the property name but cannot set the requested value.</p>

