org.xml.sax

Interface ContentHandler

All Known Subinterfaces:
TemplatesHandler, TransformerHandler

All Known Implementing Classes:
DefaultHandler, XMLFilterImpl, XMLReaderAdapter

public interface **ContentHandler**

Receive notification of the logical content of a document.

*This module, both source code and documentation, is in the Public Domain, and comes with **NO WARRANTY**.*

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 like the start and end of elements and character data.

The order of events in this interface is very 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 subelements) 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 non-validating XML processors).

Implementors should note that there is also a Java class `ContentHandler` in the java.net package; that means that it's probably a bad idea to do

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

In fact, "import ...*" is usually a sign of sloppy programming anyway, so the user should consider this a feature rather than a bug.

Since:
SAX 2.0
See Also:
`XMLReader`, `DTDHandler`, `ErrorHandler`

Method Summary

void	**characters**(char[] ch, int start, int length) Receive notification of character data.
void	**endDocument**() Receive notification of the end of a document.
void	**endElement**(String namespaceURI, String localName, String qName) Receive notification of the end of an element.
void	**endPrefixMapping**(String prefix) End the scope of a prefix-URI mapping.
void	**ignorableWhitespace**(char[] ch, int start, int length) Receive notification of ignorable whitespace in element content.
void	**processingInstruction**(String target, String data) Receive notification of a processing instruction.
void	**setDocumentLocator**(Locator locator) Receive an object for locating the origin of SAX document events.
void	**skippedEntity**(String name)

	Receive notification of a skipped entity.
void	**startDocument**() Receive notification of the beginning of a document.
void	**startElement**(String namespaceURI, String localName, String qName, Attributes atts) Receive notification of the beginning of an element.
void	**startPrefixMapping**(String prefix, String uri) Begin the scope of a prefix-URI Namespace mapping.

Method Detail

setDocumentLocator

```
public void setDocumentLocator(Locator locator)
```

Receive an object for locating the origin of SAX document events.

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 ContentHandler interface.

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 will use 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.

Note that the locator will return correct information only during the invocation of the events in this interface. The application should not attempt to use it at any other time.

Parameters:
`locator` - An object that can return the location of any SAX document event.
See Also:
`Locator`

---

startDocument

```
public void startDocument()
                  throws SAXException
```

Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in `DTDHandler` (except for `setDocumentLocator`).

Throws:
`SAXException` - Any SAX exception, possibly wrapping another exception.
See Also:
`endDocument()`

---

endDocument

```
public void endDocument()
                throws SAXException
```

Receive notification of the end of a document.

The SAX parser will invoke this method only once, and it will be the last method invoked during the parse. The parser shall not invoke this method until it has either abandoned parsing (because of an unrecoverable error) or reached the end of input.

Throws:
`SAXException` - Any SAX exception, possibly wrapping another exception.
See Also:
`startDocument()`

---

startPrefixMapping

```
public void startPrefixMapping(String prefix,
                               String uri)
                        throws SAXException
```

Begin the scope of a prefix-URI Namespace mapping.

The information from this event is not necessary for normal Namespace processing: the SAX XML reader will automatically replace prefixes for element and attribute names when the `http://xml.org/sax/features/namespaces` feature is *true* (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values, where they cannot safely be expanded automatically; the start/endPrefixMapping event supplies the information to the application to expand prefixes in those contexts itself, if necessary.

Note that start/endPrefixMapping events are not guaranteed to be properly nested relative to each-other: all startPrefixMapping events will occur before the corresponding `startElement` event, and all `endPrefixMapping` events will occur after the corresponding `endElement` event, but their order is not otherwise guaranteed.

There should never be start/endPrefixMapping events for the "xml" prefix, since it is predeclared and immutable.

Parameters:
`prefix` - The Namespace prefix being declared.
`uri` - The Namespace URI the prefix is mapped to.
Throws:
`SAXException` - The client may throw an exception during processing.
See Also:
`endPrefixMapping(java.lang.String)`, `startElement(java.lang.String, java.lang.String, java.lang.String, org.xml.sax.Attributes)`

---

endPrefixMapping

```
public void endPrefixMapping(String prefix)
                      throws SAXException
```

End the scope of a prefix-URI mapping.

See `startPrefixMapping` for details. This event will always occur after the corresponding `endElement` event, but the order of `endPrefixMapping` events is not otherwise guaranteed.

Parameters:
`prefix` - The prefix that was being mapping.
Throws:
`SAXException` - The client may throw an exception during processing.
See Also:
`startPrefixMapping(java.lang.String, java.lang.String)`, `endElement(java.lang.String, java.lang.String, java.lang.String)`

---

startElement

```
public void startElement(String namespaceURI,
                         String localName,
                         String qName,
                         Attributes atts)
                  throws SAXException
```

Receive notification of the beginning of an element.

The Parser will invoke this method at the beginning of every element in the XML document; there will be a corresponding `endElement` event for every startElement event (even when the element is empty). All of the element's content will be reported, in order, before the corresponding endElement event.

This event allows up to three name components for each element:

1. the Namespace URI;
2. the local name; and
3. the qualified (prefixed) name.

Any or all of these may be provided, depending on the values of the *http://xml.org/sax/features/namespaces* and the *http://xml.org/sax/features/namespace-prefixes* properties:

- the Namespace URI and local name are required when the namespaces property is *true* (the default), and are optional when the

namespaces property is *false* (if one is specified, both must be);
- the qualified name is required when the namespace-prefixes property is *true*, and is optional when the namespace-prefixes property is *false* (the default).

Note that the attribute list provided will contain only attributes with explicit values (specified or defaulted): #IMPLIED attributes will be omitted. The attribute list will contain attributes used for Namespace declarations (xmlns* attributes) only if the `http://xml.org/sax/features/namespace-prefixes` property is true (it is false by default, and support for a true value is optional).

Parameters:
`localName` - The local name (without prefix), or the empty string if Namespace processing is not being performed.
`qName` - The qualified name (with prefix), or the empty string if qualified names are not available.
`atts` - The attributes attached to the element. If there are no attributes, it shall be an empty Attributes object.
Throws:
`SAXException` - Any SAX exception, possibly wrapping another exception.
See Also:
`endElement(java.lang.String, java.lang.String, java.lang.String)`, `Attributes`

---

endElement

```
public void endElement(String namespaceURI,
                       String localName,
                       String qName)
                throws SAXException
```

Receive notification of the end of an element.

The SAX parser will invoke this method at the end of every element in the XML document; there will be a corresponding `startElement` event for every endElement event (even when the element is empty).

For information on the names, see startElement.

Parameters:
`localName` - The local name (without prefix), or the empty string if Namespace processing is not being performed.
`qName` - The qualified XML 1.0 name (with prefix), or the empty string if qualified names are not available.
Throws:
`SAXException` - Any SAX exception, possibly wrapping another exception.

---

characters

```
public void characters(char[] ch,
                       int start,
                       int length)
                throws SAXException
```

Receive notification of character data.

The Parser will call 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 of the characters in any single event must come from the same external entity so that the Locator provides useful information.

The application must not attempt to read from the array outside of the specified range.

Note that some parsers will report whitespace in element content using the `ignorableWhitespace` method rather than this one (validating parsers *must* do so).

Parameters:
`ch` - The characters from the XML document.
`start` - The start position in the array.
`length` - The number of characters to read from the array.
Throws:
`SAXException` - Any SAX exception, possibly wrapping another exception.
See Also:
`ignorableWhitespace(char[], int, int)`, `Locator`

---

ignorableWhitespace

```
public void ignorableWhitespace(char[] ch,
                                int start,
                                int length)
```

```
                        throws SAXException
```

Receive notification of ignorable whitespace in element content.

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): non-validating parsers may also use this method if they are capable of parsing and using content models.

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.

The application must not attempt to read from the array outside of the specified range.

Parameters:

`ch` - The characters from the XML document.
`start` - The start position in the array.
`length` - The number of characters to read from the array.

Throws:

`SAXException` - Any SAX exception, possibly wrapping another exception.

See Also:

`characters(char[], int, int)`

---

processingInstruction

```
public void processingInstruction(String target,
                                  String data)
                           throws SAXException
```

Receive notification of a processing instruction.

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.

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.

Parameters:

`target` - The processing instruction target.
`data` - The processing instruction data, or null if none was supplied. The data does not include any whitespace separating it from the target.

Throws:

`SAXException` - Any SAX exception, possibly wrapping another exception.

---

skippedEntity

```
public void skippedEntity(String name)
                   throws SAXException
```

Receive notification of a skipped entity.

The Parser will invoke this method once for each entity skipped. Non-validating 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 `http://xml.org/sax/features/external-general-entities` and the `http://xml.org/sax/features/external-parameter-entities` properties.

Parameters:

`name` - The name of the skipped entity. If it is a parameter entity, the name will begin with '%', and if it is the external DTD subset, it will be the string "[dtd]".

Throws:

`SAXException` - Any SAX exception, possibly wrapping another exception.
