Oracle®

XML Reference

10*g* (9.0.4)

Part No. B10926-01

September 2003

ORACLE®

Oracle XML Reference 10*g* (9.0.4)

Part No. B10926-01

Author: Roza Leyderman

Contents

2 Document Object Model (DOM)

3 XML Processing for Java (JAXP)

4 XSLT Processing for Java

5 XML Schema Processing

6 XML Class Generation for Java

7 XML SQL Utility for Java

8 XSQL Pages Publishing Framework for Java

9 TransX Utility for Java

10 Oracle XML JavaBeans

15 XML Schema Processor for C++

16 XML Parser for C++

17 Oracle XML Class Generator (C++)

Index

Send Us Your Comments

Oracle XML Reference 10*g* (9.0.4)

Part No. B10926-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: appserverdocs_us@oracle.com
- FAX: (650) 506-7375 Attn: Oracle Application Server Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 1op6
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This preface contains the following topics:

- Intended Audience
- Documentation Accessibility
- Organization
- Related Documentation
- Conventions

Intended Audience

Oracle XML Reference is intended for anyone with an interest in the latest Internet technologies, specifically in Oracle support for XML technology.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This document contains:

Part I, "XDK for Java Packages"

Chapters in this part cover JAVA APIs for XML.

Chapter 1, "XML Parser for Java"

Chapter 2, "Document Object Model (DOM)"

Chapter 3, "XML Processing for Java (JAXP)"

Chapter 4, "XSLT Processing for Java"

Chapter 5, "XML Schema Processing"

Chapter 6, "XML Class Generation for Java"

Chapter 7, "XML SQL Utility for Java"

Chapter 8, "XSQL Pages Publishing Framework for Java"

Chapter 9, "TransX Utility for Java"

Chapter 10, "Oracle XML JavaBeans"

Chapter 11, "Compression for Java"

Chapter 12, "Simple Object Access Protocol (SOAP)"

Part II, "C Support for XML"

Chapters in this part cover C APIs for XML.

Chapter 14, "XML Parser for C"

Chapter 13, "XML Schema Processor for C"

Part III, "C++ Support for XML"

Chapters in this part cover C++ APIs for XML.

Chapter 16, "XML Parser for C++"

Chapter 17, "Oracle XML Class Generator (C++)"

Chapter 15, "XML Schema Processor for C++"

Related Documentation

For more information, programmers should see various titles in the database documentation library.

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/membership/`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/documentation/`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width font)`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include Parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
`lowercase monospace (fixed-width font)`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase monospace (fixed-width font) italic`	Lowercase monospace italic font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`

Convention	Meaning	Example
. . .	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	`CREATE TABLE ... AS subquery;` `SELECT col1, col2, ... , coln FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Part I

XDK for Java Packages

This section contains the following chapters:

1

XML Parser for Java

XML parsing facilitates the extension of existing applications to support XML by processing XML documents and providing access to the information contained in them through a variety of APIs. This chapter contains the following sections

- DefaultXMLDocumentHandler Class
- DocumentBuilder Class
- DOMParser Class
- NodeFactory Class
- oraxml class
- SAXAttrList Class
- SAXParser Class
- XMLParseException Class
- XMLParser Class
- XMLToken Class
- XMLTokenizer Class
- NSName Class
- XMLError Class
- XMLException Class

See Also:

- *Oracle Application Developer's Guide - XML*

DefaultXMLDocumentHandler Class

This class implements the default behavior for the `XMLDocumentHandler` interface. Application writers can extend this class when they need to implement only part of the interface.

Syntax

```
public class DefaultXMLDocumentHandler implements
oracle.xml.parser.v2.XMLDocumentHandler
```

`oracle.xml.parser.v2.DefaultXMLDocumentHandler`

Table 1–1 Summary of Methods of DefaultXMLDocumentHandler

Method	Description
DefaultXMLDocumentHandler() on page 1-3	Constructs a default document.
cDATASection() on page 1-3	Receive notification of a CDATA Section.
comment() on page 1-3	Receive notification of a comment.
endDoctype() on page 1-4	Receive notification of end of the DTD.
endElement() on page 1-4	Receive notification of the end of an element.
endPrefixMapping() on page 1-4	End the scope of a prefix-URI mapping.
getHandler() on page 1-5	Get the next pipe-line node handler.
setDoctype() on page 1-5	Receive notification of DTD. Sets the DTD.
setError() on page 1-5	Receive notification of a XMLError handler.
setHandler() on page 1-6	Receive notification of a next pipe-line node handler.
setTextDecl() on page 1-6	Receives notification of a Text XML Declaration.
setXMLDecl() on page 1-6	Receives notification of an XML Declaration.
setXMLSchema() on page 1-7	Receives notification of a XMLSchema object.
skippedEntity() on page 7	Receives notification of a skipped entity.
startElement() on page 1-7	Receives notification of the beginning of an element.
startPrefixMapping() on page 1-8	Begins the scope of a prefix-URI Namespace mapping.

DefaultXMLDocumentHandler()

Constructs a default document.

Syntax

```
public  DefaultXMLDocumentHandler();
```

cDATASection()

Receive notification of a CDATA Section. The Parser will invoke this method once for each CDATA Section found. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

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

Parameter	Description
`ch`	The CDATA section characters.
`start`	The start position in the character array.
`length`	The number of characters to use from the character array.

comment()

Receives notification of a comment. The Parser will invoke this method once for each comment found: note that comment may occur before or after the main document element. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void comment( String data);
```

Parameter	Description
`data`	The comment data, or `NULL` if none is supplied.

endDoctype()

Receives notification of end of the DTD. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void endDoctype();
```

endElement()

Receives notification of the end of an element. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception. The options are described in the following table.

Syntax	Description
`public void endElement( NSName elem);`	Receives notification of the end of an element using the element.
`public void endElement( String namespaceURI, String localName, String qName);`	Receives notification of the end of an element using the namespace, the local name, and the qualified name.

Parameter	Description
elem	NSName object.
`uri`	The Namespace URI, or the empty string if the element has no Namespace URI or if Namespace processing is not being performed.
`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.

endPrefixMapping()

End the scope of a prefix-URI mapping. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void endPrefixMapping( String prefix);
```

getHandler()

Returns the next pipe-line node handler node.

Syntax

```
public XMLDocumentHandler getHandler();
```

setDoctype()

Sets the DTD so can subsequently receive notification of that DTD. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setDoctype( DTD dtd);
```

Parameter	Description
`dtd`	The DTD.

setError()

Receives notification of a XMLError handler. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setError( XMLError he);
```

Parameter	Description
`err`	The XMLError object.

setHandler()

Receive notification of a next pipe-line node handler. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setHandler( XMLDocumentHandler h);
```

Parameter	Description
`h`	The XMLDocumentHandler node.

setTextDecl()

Receive notification of a Text XML Declaration. The Parser will invoke this method once for each text XML Decl. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setTextDecl( String version,
                         String encoding);
```

Parameter	Description
`version`	The version number.
`encoding`	The encoding name, or `NULL` if not specified.

setXMLDecl()

Receive notification of an XML Declaration. The Parser will invoke this method once for XML Decl. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setXMLDecl( String version,
                        String standalone,
                        String encoding);
```

Parameter	Description
version	The version number.
standalone	The Standalone value, or NULL if not specified.
encoding	The encoding name, or NULL if not specified.

setXMLSchema()

Receive notification of a XMLSchema object. Throws org.xml.sax.SAXException, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setXMLSchema( Object s);
```

Parameter	Description
s	The XMLSchema object.

skippedEntity()

Receives notification of a skipped entity. Throws org.xml.sax.SAXException, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void skippedEntity( String name);
```

Parameter	Description
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]".

startElement()

Receives notification of the beginning of an element. Throws org.xml.sax.SAXException, which could be any SAX exception, possibly wrapping another exception. The options are described in the following table.

Syntax	Description
`public void startElement( NSName elem, SAXAttrList attrlist);`	Receives notification of the start of an element using the element.
`public void startElement( String namespaceURI, String localName, String qName, org.xml.sax.Attributes atts);`	Receives notification of the start of an element using the namespace, the local name, and the qualified name.

Parameter	Description
`elem`	NSName object.
`attlist`	SAXAttrList for the element.
`uri`	The Namespace URI, or the empty string if the element has no Namespace URI or if Namespace processing is not being performed.
`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.

startPrefixMapping()

Begin the scope of a prefix-URI Namespace mapping. Throws `org.xml.sax.SAXException`, which could possibly wrap another exception.

Syntax

```
public void startPrefixMapping( String prefix,
                                String uri);
```

Parameter	Description
`prefix`	The Namespace prefix being declared.

Parameter	Description
`uri`	The Namespace URI the prefix is mapped to.

DocumentBuilder Class

This class implements XMLDocumentHandler (deprecated) and ContentHandler to build a DOM Tree from SAX 2.0 events. XMLDocumentHandler events are supported for backward compatibility

Syntax

```
public class DocumentBuilder
  oracle.xml.parser.v2.DocumentBuilder
```

Table 1–2 Summary of Methods of DocumentBuilder

Method	Description
DocumentBuilder() on page 1-11	Creates a document builder that can be used as XMLDocumentHandler.
attributeDecl() on page 1-12	Reports an attribute type declaration.
cDATASection() on page 1-12	Receives notification of CDATA Section data inside an element.
characters() on page 1-13	Receives notification of character data inside an element.
comment() on page 1-13	Receives notification of a comment.
elementDecl() on page 1-14	Reports an element type declaration.
endCDATA() on page 1-14	Reports the end of a CDATA section.
endDoctype() on page 1-14	Receives notification of end of the DTD.
endDocument() on page 1-14	Receives notification of the end of the document.
endDTD() on page 1-15	Reports the end of DTD declarations.
endElement() on page 1-15	Receives notification of the end of an element.
endEntity() on page 1-16	Reports the end of an entity.
externalEntityDecl() on page 1-16	Reports a parsed external entity declaration.
getCurrentNode() on page 1-16	Returns the current node being build.
getDocument() on page 1-17	Gets the document being build
ignorableWhitespace() on page 1-17	Receives notification of ignorable whitespace in element content.

Table 1–2 Summary of Methods of DocumentBuilder (Cont.)

Method	Description
internalEntityDecl() on page 1-17	Reports an internal entity declaration
processingInstruction() on page 1-18	Receives notification of a processing instruction.
retainCDATASection() on page 1-18	Sets a flag to retain CDATA sections
setDebugMode() on page 1-18	Sets a flag to turn on debug information in the document
setDoctype() on page 1-19	Receives notification of DTD Sets the DTD
setDocumentLocator() on page 1-19	Receives a Locator object for document events. By default, do nothing. Application writers may override this method in a subclass if they wish to store the locator for use with other document events.
setNodeFactory() on page 1-19	Sets an optional NodeFactory to be used for creating custom DOM trees
setTextDecl() on page 1-20	Receives notification of a Text XML Declaration. The Parser will invoke this method once for each text XML Decl.
setXMLDecl() on page 1-20	Receives notification of a XML Declaration. The Parser will invoke this method once for XML Decl.
startCDATA() on page 1-20	Reports the start of a CDATA section.
startDocument() on page 1-21	Receives notification of the beginning of the document.
startDTD() on page 1-21	Reports the start of DTD declarations, if any.
startElement() on page 1-21	Receives notification of the beginning of an element.
startEntity() on page 1-22	Reports the beginning of some internal and external XML entities. All start/indemnity events must be properly nested.

DocumentBuilder()

Default constructor. Creates a document builder that can be used as XMLDocumentHandler.

Syntax

```
public  DocumentBuilder();
```

attributeDecl()

Reports an attribute type declaration. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void attributeDecl( String eName,
                           String aName,
                           String type,
                           String valueDefault,
                           String value);
```

Parameter	Description
`eName`	The name of the associated element.
`aName`	The name of the attribute.
`type`	A string representing the attribute type.
`valueDefault`	A string representing the attribute default ("#IMPLIED", "#REQUIRED", or "#FIXED") or null if none of these applies
`value`	A string representing the attribute's default value, or `NULL` if there is none.

cDATASection()

Receives notification of CDATA Section data inside an element. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception.

Syntax

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

Parameter	Description
`ch`	The CDATA characters.
`start`	The starting position in the array.
`length`	The number of characters to use from the array.

characters()

Receive notification of character data inside an element. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

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

Parameter	Description
`ch`	An array holding the character data.
`start`	The starting position in the array.
`length`	The number of characters to use from the array.

comment()

Receives notification of a comment. Reports an XML comment anywhere in the document. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception. The options are described in the following table.

Syntax	Description
`public void comment( char[] ch, int start, int length);`	Receives notification of a comment using the parameters of the comment character array.
`public void comment( String data);`	Receives notification of a comment using the comment data.

Parameter	Description
`ch`	An array holding the characters in the comment.

Parameter	Description
start	The starting position in the array.
length	The number of characters to use from the array.
data	The comment data, or NULL if none was supplied.

elementDecl()

Report an element type declaration. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void elementDecl( String name,
                         String model);
```

Parameter	Description
name	The element type name.
model	The content model as a normalized string.

endCDATA()

Reports the end of a CDATA section. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void endCDATA();
```

endDoctype()

Receives notification of end of the DTD. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void endDoctype();
```

endDocument()

Receives notification of the end of the document. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void endDocument();
```

endDTD()

Reports the end of DTD declarations. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void endDTD();
```

endElement()

Receives notification of the end of an element. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception. The options are described in the following table.

Syntax	Description
`public void endElement( NSName elem);`	Receives notification of the end of an element using the element.
`public void endElement( String namespaceURI, String localName, String qName);`	Receives notification of the end of an element using the namespace, the local name, and the qualified name.

Parameter	Description
`elem`	NSName object.
`namespaceURI`	The Namespace URI, or the empty string if the element has no Namespace URI or if Namespace processing is not being performed.
`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.

endEntity()

Reports the end of an entity. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void endEntity( String name);
```

Parameter	Description
`name`	The name of the entity that is ending.

externalEntityDecl()

Reports a parsed external entity declaration. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void externalEntityDecl( String name,
                                 String publicId,
                                 String systemId);
```

Parameter	Description
`name`	The name of the entity. If it is a parameter entity, the name will begin with '%'.
`publicId`	The declared public identifier of the entity declaration, or null if none was declared.
`systemId`	The declared system identifier of the entity.

getCurrentNode()

Returns the current XMLNode being build.

Syntax

```
public XMLNode getCurrentNode();
```

getDocument()

Returns the document being build.

Syntax

```
public XMLDocument getDocument();
```

ignorableWhitespace()

Receives notification of ignorable whitespace in element content. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

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

Parameter	Description
`ch`	The whitespace characters.
`start`	The start position in the character array.
`length`	The number of characters to use from the character array.

internalEntityDecl()

Report an internal entity declaration. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void internalEntityDecl( String name,
                                String value);
```

Parameter	Description
`name`	The name of the entity. If it is a parameter entity, the name will begin with '%'.
`value`	The replacement text of the entity.

processingInstruction()

Receives notification of a processing instruction. Throws `org.xml.sax.SAXException`, which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void processingInstruction( String target,
                                   String data);
```

Parameter	Description
`target`	The processing instruction target.
`data`	The processing instruction data, or `NULL` if none is supplied.

retainCDATASection()

Sets a flag to retain CDATA sections

Syntax

```
public void retainCDATASection( boolean flag);
```

Parameter	Description
`flag`	Determines whether CDATA sections are retained; `TRUE` for storing, `FALSE` otherwise.

setDebugMode()

Sets a flag to turn on debug information in the document.

Syntax

```
public void setDebugMode(b oolean flag);
```

Parameter	Description
`flag`	Determines whether debug info is stored; `TRUE` for storing, `FALSE` otherwise.

setDoctype()

Sets the DTD so can subsequently receive notification of that DTD. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setDoctype( DTD dtd);
```

Parameter	Description
`did`	The DTD for the document.

setDocumentLocator()

Sets the Locator so can subsequently receive notification for this Locator object for document events. By default, do nothing. Application writers may override this method in a subclass if they wish to store the locator for use with other document events.

Syntax

```
public void setDocumentLocator( org.xml.sax.Locator locator);
```

Parameter	Description
`locator`	A locator for all SAX document events.

setNodeFactory()

Sets a optional NodeFactory to be used for creating custom DOM trees.

Syntax

```
public void setNodeFactory( NodeFactory f);
```

Parameter	Description
`f`	NodeFactory/

setTextDecl()

Receive notification of a Text XML Declaration. The Parser will invoke this method once for each text XML Decl. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setTextDecl( String version,
                         String encoding);
```

Parameter	Description
`version`	The version number, or `NULL` if not specified.
`encoding`	The encoding name, or `NULL` if not specified.

setXMLDecl()

Receive notification of a XML Declaration. The Parser will invoke this method once for XML Decl. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setXMLDecl( String version,
                        String standalone,
                        String encoding);
```

Parameter	Description
`version`	The version number.
`standalone`	The standalone value, or `NULL` if not specified.
`encoding`	The encoding name, or `NULL` if not specified.

startCDATA()

Reports the start of a CDATA section. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void startCDATA();
```

startDocument()

Receive notification of the beginning of the document. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void startDocument();
```

startDTD()

Report the start of DTD declarations, if any. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void startDTD( String name,
                      String publicId,
                      String systemId);
```

Parameter	Description
`name`	The document type name.
`publicId`	The declared public identifier for the external DTD subset, or null if none was declared.
`systemId`	The declared system identifier for the external DTD subset, or null if none was declared.

startElement()

Receives notification of the beginning of an element. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception. The options are described in the following table.

Syntax	Description
`public void startElement( NSName elem, SAXAttrList attrlist);`	Receives notification of the start of an element using the element.

Syntax	Description
`public void startElement( String namespaceURI, String localName, String qName, org.xml.sax.Attributes atts);`	Receives notification of the start of an element using the namespace, the local name, and the qualified name.

Parameter	Description
`elem`	NSName object.
`attlist`	SAXAttrList for the element.
`naemspaceURI`	The Namespace URI, or the empty string if the element has no Namespace URI or if Namespace processing is not being performed.
`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.

startEntity()

Reports the beginning of some internal and external XML entities. All start/endEntity events must be properly nested. Throws `org.xml.sax.SAXException,` which could be any SAX exception, possibly wrapping another exception.

Syntax

```
public void startEntity( String name);
```

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

DOMParser Class

This class implements an eXtensible Markup Language (XML) 1.0 parser according to the World Wide Web Consortium (W3C) recommendation. to parse a XML document and build a DOM tree.

Syntax

```
public class DOMParser
   oracle.xml.parser.v2.DOMParser
```

Table 1–3 Fields of DOMParser

Field	Syntax	Description
DEBUG_MODE	public static final java.lang.String DEBUG_MODE	Sets Debug Mode Boolean.TRUE or Boolean.FALSE
ERROR_ENCODING	public static final java.lang.String ERROR_ENCODING	Encoding for errors report through error stream (only if ERROR_STREAM is set).
ERROR_STREAM	public static final java.lang.String ERROR_STREAM	Error stream for reporting errors. The object can be OutputStream or PrintWriter. This attribute is ignored if ErrorHandler is set.
NODE_FACTORY	public static final java.lang.String NODE_FACTORY	Set NodeFactory to build custom Nodes
SHOW_WARNINGS	public static final java.lang.String SHOW_WARNINGS	Boolean to ignore warnings Boolean.TRUE or Boolean.FALSE

Table 1–4 Summary of Methods of DOMParser

Method	Description
DOMParser() on page 1-24	Creates a new parser object.
getAttribute() on page 1-24	Returns specific attributes of the underlying implementation.
getDoctype() on page 1-24	Gets the DTD.

Table 1–4 (Cont.) Summary of Methods of DOMParser

Method	Description
getDocument() on page 1-25	Gets the document.
parseDTD() on page 1-25	Parses the XML External DTD from given input source.
reset() on page 1-26	Resets the parser state
retainCDATASection() on page 1-26	Switches to determine whether to retain CDATA sections
setAttribute() on page 1-26	Sets specific attributes on the underlying implementation.
setDebugMode() on page 1-27	Sets a flag to turn on debug information in the document
setErrorStream() on page 1-27	Creates an output stream for errors and warnings.
setNodeFactory() on page 1-28	Sets the node factory.
showWarnings() on page 1-28	Switches to determine whether to print warnings

DOMParser()

Creates a new parser object.

Syntax;

```
public  DOMParser();
```

getAttribute()

Returns specific attributes on the underlying implementation. Throws `IllegalArgumentException` if the underlying implementation doesn't recognize the attribute.

Syntax

```
public java.lang.Object getAttribute( String name);
```

Parameter	Description
`name`	The name of the attribute.

getDoctype()

Returns the DTD.

Syntax

```
public DTD getDoctype();
```

getDocument()

Returns the document being parsed.

Syntax

```
public XMLDocument getDocument();
```

parseDTD()

Parses the XML External DTD from given input stream. Throws the following exceptions:

- `XMLParseException` if syntax or other error encountered.
- `SAXException`, which could be SAX exception, possibly wrapping another exception.
- `IOException` in cases of I/O Error.

The options are described in the following table.

Syntax	Description
`public final void parseDTD( org.xml.sax.InputSource in, String rootName);`	Parses the XML External DTD from an input source.
`public final void parseDTD( java.io.InputStream in, String rootName);`	Parses the XML External DTD from an input stream. The base URL should be set for resolving external entities and DTD.
`public final void parseDTD( java.io.Reader r, String rootName);`	Parses the XML External DTD from a reader. The base URL should be set for resolving external entities and DTD.
`public final void parseDTD( String in, String rootName);`	Parses the XML External DTD from a string.
`public final void parseDTD( java.net.URL url, String rootName);`	Parses the XML External DTD document poi9nted to by the given URL and creates the corresponding XML document hierarchy.

Parameter	Description
in	The input to parse.
rootName	The element to be used as root Element.
r	The reader containing XML data to parse.
url	The url which points to the XML document to parse.

reset()

Resets the parser state.

Syntax

```
public void reset();
```

retainCDATASection()

Switches to determine whether to retain CDATA sections.

Syntax

```
public void retainCDATASection( boolean flag);
```

Parameter	Description
flag	TRUE - keep CDATASections (default) FALSE - convert CDATASection to Text nodes

setAttribute()

Sets specific attributes on the underlying implementation. Throws an `IllegalArgumentException` if the underlying implementation doesn't recognize the attribute.

Syntax

```
public void setAttribute( String name,
                          Object value);
```

Parameter	Description
name	The name of the attribute.

Parameter	Description
value	he value of the attribute.

setDebugMode()

Sets a flag to turn on debug information in the document

Syntax

```
public void setDebugMode( boolean flag);
```

Parameter	Description
flag	Determines whether debug info is stored.

setErrorStream()

Creates an output stream for the output of errors and warnings. If an output stream for errors is not specified, the parser will use the standard error output stream `System.err` for outputting errors and warnings. Additionally, an exception is thrown if the encoding specified is unsupported. The options are described in the following table.

Syntax	Description
`public final void setErrorStream( java.io.OutputStream out);`	Creates an output stream for the output of errors and warnings.
`public final void setErrorStream( java.io.OutputStream out, String enc);`	Creates an output stream for the output of errors and warnings. Throws an `IOEexception` if the encoding specified is unsupported.
`public final void setErrorStream( java.io.PrintWriter out);`	Creates a Print Writer for the output of errors and warnings. Throws `IOException` if I/O error occurs in setting the error stream.

Parameter	Description
out	Used for output of errors and warnings.
enc	The encoding to use.

setNodeFactory()

Set the node factory. Applications can extend the NodeFactory and register it through this method. The parser will then use the user supplied NodeFactory to create nodes of the DOM tree. Throws `XMLParseException` if an invalid factory is set

Syntax

```
public void setNodeFactory( NodeFactory factory);
```

Parameter	Description
`factory`	The NodeFactory to set.

showWarnings()

Switches to determine whether to print warnings.

Syntax

```
public void showWarnings( boolean flag);
```

Parameter	Description
`flag`	Determines whether warnings should be shown.

NodeFactory Class

This class specifies methods to create various nodes of the DOM tree built during parsing. Applications can override these methods to create their own custom classes to be added to the DOM tree while parsing. Applications have to register their own NodeFactory using the XMLParser's setNodeFactory() method. If a null pointer is returned by these methods, then the node will not be added to the DOM tree.

Syntax

```
public class NodeFactory extends java.lang.Object implements
java.io.Serializable
```

Table 1–5 Summary of Methods of NodeFactory

Method	Description
NodeFactory() on page 1-30	Class constructor.
createAttribute() on page 1-30	Creates and returns an attribute node with the specified tag, and text.
createCDATASection() on page 1-31	Creates and returns a CDATA node with the specified text.
createComment() on page 1-31	Creates and returns a comment node with the specified text.
createDocument() on page 1-31	Creates and returns a document node. This method cannot return a null pointer.
createDocumentFragment() on page 1-31	Creates and returns a document fragment node with the specified tag.
createElement() on page 1-32	Creates and returns an Element node with the specified tag.
createElementNS() on page 1-32	Creates and returns an Element node with the specified local name, prefix, namespaceURI
createEntityReference() on page 1-32	Creates and returns an entity reference node with the specified tag.
createProcessingInstruction() on page 1-33	Creates and returns a PI node with the specified tag, and text.

Table 1–5 (Cont.) Summary of Methods of NodeFactory

Method	Description
createTextNode() on page 1-33	Creates and returns s a text node with the specified text.

NodeFactory()

Class constructor.

Syntax

```
public  NodeFactory();
```

createAttribute()

Creates and returns an attribute node. The options are described in the following table.

Syntax	Description
`public XMLAttr createAttribute( String tag, String text);`	Creates and returns an attribute node with the specified tag and text.
`public XMLAttr createAttribute( String localName, String prefix, String namespaceURI, String value);`	Creates and returns an attribute node with the specified local name, prefix, namespaceURI, and value.

Parameter	Description
`tag`	The name of the node.
`text`	The text associated with the node.
`localName`	The name of the node.
`prefix`	The prefix of the node.
`naemspaceURI`	The namespace of the node.
`value`	The value associated with the node.

createCDATASection()

Creates and returns CDATA node with the specified text.

Syntax

```
public XMLCDATA createCDATASection( String text);
```

Parameter	Description
text	The text associated with the node.

createComment()

Creates and returns a comment node with the specified text.

Syntax

```
public XMLComment createComment( String text);
```

Parameter	Description
text	The text associated with the node.

createDocument()

Creates and returns a document node. This method cannot return a null pointer.

Syntax

```
public XMLDocument createDocument();
```

createDocumentFragment()

Creates a document fragment node with the specified tag.

Syntax

```
public XMLDocumentFragment createDocumentFragment();
```

Returns

The created document fragment node.

createElement()

Creates and returns an Element node with the specified tag.

Syntax

```
public XMLElement createElement( String tag);
```

Parameter	Description
tag	The name of the element.

createElementNS()

Creates and returns an Element node with the specified local name, prefix, and namespaceURI.

Syntax

```
public XMLElement createElementNS( String localName,
                                   String prefix,
                                   String namespaceURI);
```

Parameters	Description
localName	The name of the element.
prefix	The prefix of an element.
namespaceURI	The namespace of the element.

createEntityReference()

Creates and returns an entity reference node with the specified tag.

Syntax

```
public XMLEntityReference createEntityReference( String tag);
```

Parameter	Description
tag	The name of the node.

createProcessingInstruction()

Creates and returns a ProcessingInstruction node with the specified tag, and text.

Syntax

```
public XMLPI createProcessingInstruction( String tag,
                                          String text);
```

Parameter	Description
tag	The name of the node.
text	The text associated with the node.

createTextNode()

Creates and returns a text node with the specified text.

Syntax

```
public XMLText createTextNode( String text);
```

Parameter	Description
text	The text associated with the node.

oraxml class

The `oraxml` class provides a command-line interface to validate XML files.

Table 1–6 Command-line interface of oraxml

command	description
`-help`	Prints the help message.
`-version`	Prints the release version.
`-novalidate`	Parses the input file to check for well-formedness.
`-dtd`	Validates the input file with DTD validation.
`-schema`	Validates the input file with Schema validation.
`-log <logfile>`	Writes the errors/logs to the output file.
`-comp`	Compresses the input xml file.
`-decomp`	Decompresses the input compressed file.
`-enc`	Prints the encoding of the input file.
`-warning`	Show warnings.

Table 1–7 Summary of Methods of oraxml

Method	Description
oraxml() on page 1-34	Class constructor.
main() on page 1-35	Operation loop.

oraxml()

Syntax

```
public oraxml();
```

main()

Syntax

```
public static void main( String[] args);
```

SAXAttrList Class

This class implements the SAX `AttributeList` interface and also provides Namespace support. Applications that require Namespace support can explicitly cast any attribute list returned by an Oracle parser class to `SAXAttrList` and use the methods described here. It also implements Attributes (SAX 2.0) interface.

This interface allows access to a list of attributes in three different ways:

- by attribute index;
- by Namespace-qualified name; or
- by qualified (prefixed) name.

This interface replaces the now-deprecated SAX1 interface, which does not contain Namespace support. In addition to Namespace support, it adds the getIndex methods.

The order of attributes in the list is unspecified, and will vary from implementation to implementation.

Syntax

```
public class SAXAttrList
   oracle.xml.parser.v2.SAXAttrList
```

Table 1–8 Summary of Methods of SAXAttrList

Method	Description
SAXAttrList() on page 1-37	Class constructor.
addAttr() on page 1-37	Adds an attribute to the parent element node.
getExpandedName() on page 1-38	Returns the expanded name for an attribute in the list (by position).
getIndex() on page 1-38	Returns the index of an attribute.
getLength() on page 1-39	Returns the number of attributes in the list.
getLocalName() on page 1-39	Returns an attribute's local name by index.
getPrefix() on page 1-39	Returns the namesp.ace prefix for an attribute in the list by position.

Table 1–8 (Cont.) Summary of Methods of SAXAttrList

Method	Description
getQName() on page 1-40	Returns an attribute's qualified name by index.
getType() on page 1-40	Returns an attribute's type.
getURI() on page 1-41	Returns an attribute's Namespace URI by index
getValue() on page 1-41	Returns the attribute's value as a string,
reset() on page 1-42	Resets the SAXAttrList.

SAXAttrList()

Class constructor.

Syntax

```
public  SAXAttrList(int elems)
```

addAttr()

Adds an attribute to the parent element node. The options are described in the following table.

Syntax	Description
`public void addAttr( String pfx, String lname, String tag, String value, boolean spec, int type);`	Adds attribute using prefix, local name, qualified name, value, specified flag, and attribute type.
`public void addAttr( String pfx, String lname, String tag, String value, boolean spec, int type, String nmsp);`	Adds attribute using prefix, local name, qualified name, value, specified flag, attribute type, and namespace.

Parameter	Description
pfx	The prefix of the attribute.
lname	The local name of the attribute.
tag	The qname of the attribute.
value	The attribute value.
spec	The specified flag.
type	The attribute type.
nmsp	The namespace.

getExpandedName()

Returns the expanded name for an attribute in the list (by position).

Syntax

```
public String getExpandedName( int i);
```

Parameter	Description
i	The index of the attribute in the list.

getIndex()

Returns the index of an attribute. Returns `-1` if it does not appear in the list. The options are described in the following table.

Syntax	Description
`public int getIndex( String qName);`	Returns the index of an attribute by qualified name.
`public int getIndex( String uri, String localName);`	Returns the index of an attribute by namespace URI and local name.

Parameter	Description
qName	The qualified (prefixed) name.
uri	The Namespace URI, or the empty string if the name has no Namespace URI.
localName	The attribute's local name.

getLength()

Returns the number of attributes in this list. The SAX parser may provide attributes in any arbitrary order, regardless of the order in which they were declared or specified. The number of attributes may be zero.

Syntax

```
public int getLength();
```

getLocalName()

Returns an attribute's local name by index, or the empty string if Namespace processing is not being performed, or NULL if the index is out of range.

Syntax

```
public String getLocalName( int index);
```

Parameter	Description
index	The attribute index (zero-based).

getPrefix()

Returns the namespace prefix for an attribute in the list by position.

Syntax

```
public String getPrefix( int index);
```

Parameter	Description
index	The attribute index (zero-based).

getQName()

Returns an attribute's XML 1.0 qualified name by index, or the empty string if none is available, or NULL if the index is out of range.

Syntax

```
public String getQName( int index);
```

Parameter	Description
index	The attribute index (zero-based).

getType()

Returns an attribute's type. The attribute type is one of the strings "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES", or "NOTATION" (always in upper case). 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"). For an enumerated attribute that is not a notation, the parser will report the type as "NMTOKEN". The options are described in the following table.

Syntax	Description
`public String getType( int index);`	Returns an attribute's type by index, or NULL if the index is out of range.
`public String getType( String qName);`	Returns an attribute's type by qualified name, or returns NULL if the attribute is not in the list or if qualified names are not available.
`public String getType( String uri, String localName);`	Returns an attribute's type by Namespace name, or returns NULL if the attribute is not in the list or if Namespace processing is not being performed.

Parameter	Description
index	The attribute index (zero-based).

Parameter	Description
qName.	The XML 1.0 qualified name.
uri	The Namespace URI, or the empty String if the name has no Namespace URI.
localName	The local name of the attribute.

getURI()

Returns an attribute's Namespace URI by index, or the empty string if none is available, or null if the index is out of range.

Syntax

```
public String getURI( int index);
```

Parameter	Description
index	The attribute index (zero-based).

getValue()

Returns the attribute's value as a string, or NULL if the index is out of range. If the attribute value is a list of tokens (IDREFS, ENTITIES, or NMTOKENS), the tokens will be concatenated into a single string with each token separated by a single space. The options are described in the following table.

Syntax	Description
`public String getValue( int index);`	Returns the attribute's value by index.
`public String getValue( String qName);`	Returns the attribute's value by qualified name.
`public String getValue( String uri, String localName);`	Returns the attribute's value by Namespace name.

Parameter	Description
index	The attribute index (zero-based).
qName	The XML 1.0 qualified name.
uri	The Namespace URI, or the empty String if the name has no Namespace URI.
localName	The local name of the attribute.

reset()

Resets the SAXAttrList.

Syntax

```
public void reset();
```

SAXParser Class

This class implements an eXtensible Markup Language (XML) 1.0 SAX parser according to the World Wide Web Consortium (W3C) recommendation. Applications can register a SAX handler to receive notification of various parser events.

XMLReader is the interface that an XML parser's SAX2 driver must implement. This interface allows 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:

- it adds a standard way to query and set features and properties; and
- it adds Namespace support, which is required for many higher-level XML standards.

Syntax

```
public class SAXParser
  oracle.xml.parser.v2.SAXParser
```

Table 1–9 Summary of Methods of SAXParser

Method	Description
SAXParser() on page 1-44	Creates a new parser object.
getContentHandler() on page 1-44	Returns the current content handler.
getDTDHandler() on page 1-44	Returns the current DTD handler.
getFeature() on page 1-44	Returns the current state of the feature (true or false).
getProperty() on page 1-45	Returns the value of a property.
setContentHandler() on page 1-46	Registers a content event handler.

Table 1–9 (Cont.) Summary of Methods of SAXParser

Method	Description
setDTDHandler() on page 1-46	Registers a DTD event handler.
setFeature() on page 1-46	Set the state of a feature.
setProperty() on page 1-47	Sets the value of a property.

SAXParser()

Creates a new parser object.

Syntax

```
public  SAXParser()
```

getContentHandler()

Returns the current content handler, or null if none has been registered.

Syntax

```
public org.xml.sax.ContentHandler getContentHandler();
```

getDTDHandler()

Returns the current DTD handler, or null if none has been registered.

Syntax

```
public org.xml.sax.DTDHandler getDTDHandler();
```

getFeature()

Returns the current state of the feature (true or false). The feature name is any fully-qualified URI. It is possible for an XMLReader to recognize a feature name but not be able to return its value; this is especially true in the case of an adapter for a SAX1 Parser, which has no way of knowing whether the underlying parser is performing validation or expanding external entities. Some feature values may be available only in specific contexts, such as before, during, or after a parse. Throws the following exceptions:

- `org.xml.sax.SAXNotRecognizedException` if the XMLReader does not recognize the feature name.
- `org.xml.sax.SAXNotSupportedException` if the XMLReader recognizes the feature name but cannot determine its value at this time.

Due to its binary input value, the feature only controls DTD validation. The value true sets DTD validation to TRUE. This feature cannot be used to control XML Schema based validation.

Implementors are free (and encouraged) to invent their own features, using names built on their own URIs.

Syntax

```
public boolean getFeature( String name);
```

Parameter	Description
name	Feature name.

getProperty()

Returns the value of a property. The property name is any fully-qualified URI. It is possible for an XMLReader to recognize a property name but to be unable to return its state; this is especially true in the case of an adapter for a SAX1 `Parser`. XMLReaders are not required to recognize any specific property names, though an initial core set is documented for SAX2. Some property values may be available only in specific contexts, such as before, during, or after a parse. Implementors are free (and encouraged) to invent their own properties, using names built on their own URIs. Throws the following exceptions:

- `org.xml.sax.SAXNotRecognizedException` if the XMLReader does not recognize the feature name.
- `org.xml.sax.SAXNotSupportedException` if the XMLReader recognizes the feature name but cannot determine its value at this time.

Syntax

```
public Object getProperty( String name);
```

Parameter	Description
`name`	The property name, which is a fully-qualified URI

setContentHandler()

Registers a content event handler. If the application does not register a content handler, all content events reported by the SAX parser will be 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. Throws `java.lang.NullPointerException` if the handler argument is null.

Syntax

```
public void setContentHandler( org.xml.sax.ContentHandler handler);
```

Parameter	Description
`handler`	The content handler.

setDTDHandler()

Registers a DTD event handler. If the application does not register a DTD handler, all DTD events reported by the SAX parser will be 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. If the handler argument is null, throws `java.lang.NullPointerException`.

Syntax

```
public void setDTDHandler( org.xml.sax.DTDHandler handler);
```

Parameter	Description
`handler`	The DTD handler.

setFeature()

Set the state of a feature. The feature name is any fully-qualified URI. It is possible for an XMLReader to recognize a feature name but to be unable to set its value; this is especially true in the case of an adapter for a SAX1 `Parser`, which has no way of affecting whether the underlying parser is validating, for example. Some feature values may be immutable or mutable only in specific contexts, such as before, during, or after a parse. Due to its binary input value, the feature "http://www.xml.org/sax/features/validation" only controls DTD validation. The

value true sets DTD validation to TRUE. This feature cannot be used to control XML Schema based validation. Throws the following exceptions:

- `org.xml.sax.SAXNotRecognizedException` - When the XMLReader does not recognize the feature name.
- `org.xml.sax.SAXNotSupportedException` - When the XMLReader recognizes the feature name but cannot set the requested value.

Syntax

```
public void setFeature( String name, boolean value);
```

Parameter	Description
`name`	The feature name, which is a fully-qualified URI.
`state`	The requested state of the feature (true or false).

setProperty()

Sets the value of a property. The property name is any fully-qualified URI. It is possible for an XMLReader to recognize a property name but to be unable to set its value; this is especially true in the case of an adapter for a SAX1 `Parser`. XMLReaders are not required to recognize setting any specific property names, though a core set is provided with SAX2. Some property values may be immutable or mutable only in specific contexts, such as before, during, or after a parse. This method is also the standard mechanism for setting extended handlers. Throws the following exceptions:

- `org.xml.sax.SAXNotRecognizedException` if the XMLReader does not recognize the property name.
- `org.xml.sax.SAXNotSupportedException` if the XMLReader recognizes the property name but cannot set the requested value.

Syntax

```
public void setProperty( String name, Object value);
```

Parameter	Description
`name`	The property name, which is a fully-qualified URI.
`state`	The requested value for the property.

XMLParseException Class

Indicates that a parsing exception occurred while processing an XML document

Syntax

```
public class XMLParseException
   oracle.xml.parser.v2.XMLParseException
```

Table 1–10 Fields of XMLParseException

Field	Syntax	Description
ERROR	public static final int ERROR	Code for non-fatal error
FATAL_ERROR	public static final int FATAL_ERROR	Code for fatal error
WARNING	public static final int WARNING	Code for warning

Table 1–11 Summary of Methods of XMLParseException

Method	Description
XMLParseException() on page 1-49	Constructor for the XMLParse Exception class.
formatErrorMessage() on page 1-49	Formats error message at specified index.
getColumnNumber() on page 1-49	Returns column number of error at specified index.
getException() on page 1-50	Returns the exception that occurred in error at specified index.
getLineNumber() on page 1-50	Returns the line number of error at specified index.
getMessage() on page 1-50	Returns the error message at specified index.
getMessageType() on page 1-51	Returns the type of message at specified index.
getNumMessages() on page 1-51	Returns the total number of errors/warnings found during parsing.
getPublicId() on page 1-51	Returns the public id of input when error at specific index occurred.
getSystemId() on page 1-51	Returns the system ID of input when error at specific index occurred.

XMLParseException()

Class constructor.

Syntax

```
public  XMLParseException( String mesg,
                           String pubId,
                           String sysId,
                           int line,
                           int col,
                           int type);
```

Parameter	Description
mesg	Message.
pubId	Public id.
sysId	System id.
line	Line.
col	Column.
type	Type.

formatErrorMessage()

Returns the error message at specified index.

Syntax

```
public String formatErrorMessage( int i);
```

Parameter	Description
i	Index.

getColumnNumber()

Returns the column number of error at specified index.

Syntax

```
public int getColumnNumber(int i);
```

Parameter	Description
i	Index.

getException()

Returns the exception (if exists) that occurred in error at specified index.

Syntax

```
public java.lang.Exception getException(int i);
```

Parameter	Description
i	Index.

getLineNumber()

Returns the line number of error at specified index

Syntax

```
public int getLineNumber(int i);
```

Parameter	Description
i	Index.

getMessage()

Returns the error message at specified index

Syntax

```
public String getMessage(int i)
```

Parameter	Description
i	Index.

getMessageType()

Returns the type of the error message at specified index.

Syntax

```
public int getMessageType(int i)
```

Parameter	Description
i	Index.

getNumMessages()

Return the total number of errors/warnings found during parsing.

Syntax

```
public int getNumMessages();
```

getPublicId()

Returns the public ID of input when error at specified index occurred.

Syntax

```
public String getPublicId(int i);
```

Parameter	Description
i	Index

getSystemId()

Returns the system ID of input when error at specified index occurred.

Syntax

```
public String getSystemId(int i);
```

Parameter	Description
i	Index

XMLParser Class

This class serves as a base class for the `DOMParser` and `SAXParser` classes. It contains methods to parse eXtensible Markup Language (XML) 1.0 documents according to the World Wide Web Consortium (W3C) recommendation. This class cannot be instantiated (applications may use the DOM or SAX parser depending on their requirements).

Syntae

```
public abstract class XMLParser
   oracle.xml.parser.v2.XMLParser
```

Table 1–12 Fields of XMLParser

Field	Syntax	Description
BASE_URL	public static final java.lang.String BASE_URL	Base URL used in parsing entities. Replaces setBaseURL(); Should be URL object.
DTD_OBJECT	public static final java.lang.String DTD_OBJECT	DTD Object to be used for validation. Replaces XMLParser.setDoctype().
SCHEMA_ OBJECT	public static final java.lang.String SCHEMA_OBJECT	Schema Object to be used for validation. Replaces XMLParser.setXMLSchema().
STANDALONE	public static final java.lang.String STANDALONE	Sets the standalone property of the input files. If true the DTDs are not retrieved.
USE_DTD_ ONLY_FOR_ VALIDATION	public static final java.lang.String USE_DTD_ONLY_FOR_VALIDATION	If true, DTD Object is used only for validation and is not added to the parser document (Boolean.TRUE or Boolean.FALSE) This property/attribute is only if setDoctype is called to use a fixed DTD.

Methods of XMLParser

Table 1–13 Summary of Methods of XMLParser

Method	Description
getAttribute() on page 1-55	Retrieves value of attribute of the implementation.
getBaseURL() on page 1-55	Returns the base URL.
getEntityResolver() on page 1-55	Returns the entity resolver.
getErrorHandler() on page 1-55	Returns the error handler.
getReleaseVersion() on page 1-56	Returns the release version.
getValidationModeValue() on page 1-56	Returns the validation mode value.
getXMLProperty() on page 1-56	Returns the value of a property.
isXMLPropertyReadOnly() on page 1-56	Returns `TRUE` if a given property is read-only.
isXMLPropertySupported( ) on page 1-57	Returns `TRUE` if a given property is supported.
parse() on page 1-57	Parses the XML.
reset() on page 1-58	Resets the parser state.
setAttribute() on page 1-58	Sets specific attributes on the underlying implementation.
setBaseURL() on page 1-58	Sets the base URL for loading external entities and DTDs.
setDoctype() on page 1-59	Sets the DTD.
setEntityResolver() on page 1-59	Sets the entity resolver.
setErrorHandler() on page 1-59	Registers an error event handler.
setLocale() on page 1-60	Sets the locale for error reporting.
setPreserveWhitespace() on page 1-60	Sets the white space preserving mode.
setValidationMode() on page 1-61	Sets the validation mode.

Table 1–13 (Cont.) Summary of Methods of XMLParser

Method	Description
setXMLProperty() on page 1-61	Sets and returns an XMLProperty.
setXMLSchema() on page 1-61	Sets an XML Schema for validating the instance document.

getAttribute()

Retrieves values of specific attributes on the underlying implementation. Throws `IllegalArgumentException` - thrown if the underlying implementation doesn't recognize the attribute.

Syntax

```
public java.lang.Object getAttribute( String name);
```

Parameter	Description
`value`	The value of the attribute.

getBaseURL()

Returns the base URL.

Syntax

```
public java.net.URL getBaseURL();
```

getEntityResolver()

Returns the current entity resolver.

Syntax

```
public org.xml.sax.EntityResolver getEntityResolver();
```

Returns

The current entity resolver, or null if none has been registered.

getErrorHandler()

Returns the current error handler, or null if none has been registered.

Syntax;

```
public org.xml.sax.ErrorHandler getErrorHandler();
```

getReleaseVersion()

Returns the release version of the Oracle XML Parser.

Syntax

```
public static String getReleaseVersion();
```

getValidationModeValue()

Returns the validation mode value:

- 0 if the XML parser is NONVALIDATING
- 1 if the XML parser is PARTIAL_VALIDATION
- 2 if the XML parser is DTD_VALIDATION
- 3 if the XML parser is SCHEMA_VALIDATION

Syntax

```
public int getValidationModeValue();
```

getXMLProperty()

Returns value of a property. The property is returned if present and supported, else null is returned

Syntax

```
public java.lang.Object getXMLProperty( String name);
```

Parameter	Description
name	Name of the property.

isXMLPropertyReadOnly()

Returns TRUE if a given property is read-only Returns true if the property is not supported

Syntax

```
public boolean isXMLPropertyReadOnly( String name);
```

Parameter	Description
name	Name of the property.

isXMLPropertySupported()

Returns TRUE if a given property is supported.

Syntax

```
public boolean isXMLPropertySupported( String name)
```

Parameter	Description
name	Name of the property.

parse()

Parses the XML. Throws the following exceptions:

- XMLParseException if syntax or other error encountered.
- SAXException, which could be any SAX exception, possibly wrapping another exception.
- IOException in cases of I/O Error.

The options are described in the following table.

Syntax	Description
public void parse(org.xml.sax.InputSource in);	Parses the XML from given input source.
public final void parse(java.io.InputStream in);	Parses the XML from given input stream.
public final void parse(java.io.Reader in);	Parses the XML from given reader.
public void parse(String in);	Parses the XML from the URL indicated.

Syntax	Description
`public final void parse( java.net.URL url);`	Parses the XML document pointed to by the given URL and creates the corresponding XML document hierarchy.

Parameter	Description
`in`	Source containing XML data to parse.
`url`	The ULR pointing to the XML document which will be parsed.

reset()

Resets the parser state.

Syntax

```
public void reset();
```

setAttribute()

Sets specific attributes on the underlying implementation. Throws an `IllegalArgumentException` if the underlying implementation doesn't recognize the attribute.

Syntax

```
public void setAttribute( String name,
                          Object value);
```

Parameter	Description
`name`	The name of the attribute.
`value`	The value of the attribute.

setBaseURL()

Sets the base URL for loading external entities and DTDs. This method should to be called if the parse() is used to parse the XML Document.

Syntax

```
public void setBaseURL( java.net.URL url);
```

Parameter	Description
`url`	The base URL.

setDoctype()

Sets the DTD.

Syntax

```
public void setDoctype( DTD dtd);
```

Parameter	Description
`dtd`	The DTD to set and use while parsing.

setEntityResolver()

Registers an entity resolver. Throws a `NullPointerException` if the resolver argument is null. If the application does not register an entity resolver, the XMLReader 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.

Syntax

```
public void setEntityResolver( org.xml.sax.EntityResolver resolver);
```

Parameter	Description
resolver	The entity resolver.

setErrorHandler()

Registers an error event handler. Throws `java.lang.NullPointerException` if the handler argument is null. If the application does not register an error handler, all error events reported by the SAX parser will be silently ignored; however, normal

processing may not continue. It is highly recommended that all SAX applications implement an error handler to avoid unexpected bugs. 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.

Syntax

```
public void setErrorHandler( org.xml.sax.ErrorHandler handler);
```

Parameter	Description
handler	The error handler.

Parameters

`handler` - The error handler.

setLocale()

Sets the locale for error reporting. Throws a `SAXException` on error.

Syntax

```
public void setLocale( java.util.Locale locale);
```

Parameter	Description
locale	The locale to set.

setPreserveWhitespace()

Sets the white space preserving mode.

Syntax

```
public void setPreserveWhitespace( boolean flag);
```

Parameter	Description
flag	The preserving mode.

setValidationMode()

Sets the validation mode. This method sets the validation mode of the parser to one of the 4 types: NONVALIDATING, PARTIAL_VALIDATION, DTD_VALIDATION and SCHEMA_VALIDATION.

Syntax

```
public void setValidationMode(int valMode)
```

Parameter	Description
valMode	Determines the type of the validation mode to which the parser should be set.

setXMLProperty()

Sets and returns an XMLproperty. The value of the property set is returned if successfully set, a null is returned if the property is read-only and cannot be set or is not supported.

Syntax

```
public java.lang.Object setXMLProperty( String name,
                                        Object value);
```

Parameter	Description
name	Name of the property.
value	Value of the property.

setXMLSchema()

Sets an XMLSchema for validating the instance document.

Syntax

```
public void setXMLSchema( java.lang.Object schema);
```

Parameter	Description
schema	The XMLSchema object

XMLToken Class

Basic interface for XMLToken. All XMLParser applications with Tokenizer feature must implement this interface. The interface has to be registered using `XMLParser` method `setTokenHandler()`.

If XMLtoken handler isn't null, then for each registered and found token the parser calls the XMLToken call-back method `token()`. During tokenizing the parser doesn't validate the document and doesn't include/read internal/external entities. If XMLtoken handler == null then the parser parses as usual.

A request for XML token is registered (on/off) using XMLParser method `setToken()`. The requests could be registered during the parsing (from inside the call-back method) as well.

The XML tokens are defined as public constants in `XMLToken` interface. They correspond to the XML syntax variables from W3C XML Syntax Specification.

Syntax

```
public interface XMLToken
```

Table 1–14 Fields of XMLToken

Field	Syntax	Description
AttListDecl	public static final int AttListDecl	AttListDecl ::= '<' '!' 'ATTLIST' S Name AttDef* S? '>'
AttName	public static final int AttName	AttName ::= Name
Attribute	public static final int Attribute	Attribute ::= AttName Eq AttValue
AttValue	public static final int AttValue	AttValue ::= '"' ([^<&"] | Reference)* '"' | "'" ([^<&'] | Reference)* "'"
CDSect	public static final int CDSect	CDSect ::= CDStart CData CDEnd CDStart ::= '<' '!' '[CDATA[' CData ::= (Char* - (Char* ']]>' Char*)) CDEnd ::= ']]>'
CharData	public static final int CharData	CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
Comment	public static final int Comment	Comment ::= '<' '!' '--' ((Char - '-') | ('-' (Char - '-')))* '-->'

Table 1–14 (Cont.) Fields of XMLToken

Field	Syntax	Description
DTDName	public static final int DTDName	DTDName ::= name
ElemDeclName	public static final int ElemDeclName	ElemDeclName ::= name
elementdecl	public static final int elementdecl	elementdecl ::= '<' '!ELEMENT' S ElemDeclName S contentspec S? '>'
EmptyElemTag	public static final int EmptyElemTag	EmptyElemTag ::= '<' STagName (S Attribute)* S? '/' '>'
EntityDecl	public static final int EntityDecl	EntityDecl ::= '<' '!' ENTITY' S EntityDeclName S EntityDef S? '>' | '<' '!' ENTITY' S '%' S EntityDeclName S PEDef S? '>' EntityDef ::= EntityValue | (ExternalID NDataDecl?) PEDef ::= EntityValue | ExternalID
EntityDeclName	public static final int EntityDeclName	EntityValue ::= '"' ([^%&"] | PEReference | Reference)* '"'| "'" ([^%&']| PEReference | Reference)* "'"
EntityValue	public static final int EntityValue	EntityDeclName ::= Name
ETag	public static final int ETag	ETag ::= '<' '/' ETagName S? '>'
ETagName	public static final int ETagName	ETagName ::= Name
ExternalID	public static final int ExternalID	ExternalID ::= 'SYSTEM' S SystemLiteral | 'PUBLIC' S PubidLiteral S SystemLiteral
NotationDecl	public static final int NotationDecl	NotationDecl ::= '<' '!NOTATION' S Name S (ExternalID | PublicID) S? '>'
PI	public static final int PI	PI ::= '<' '?' PITarget (S (Char* - (Char* '?>' Char*)))? '?' '>'
PITarget	public static final int PITarget	PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
Reference	public static final int Reference	Reference ::= EntityRef | CharRef | PEReference EntityRef ::= '&' Name ';' PEReference ::= '%' Name ';' CharRef ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'

Table 1–14 (Cont.) Fields of XMLToken

Field	Syntax	Description
STag	public static final int STag	STag ::= '<' STagName (S Attribute)* S? '>'
STagName	public static final int STagName	STagName ::= Name
TextDecl	public static final int TextDecl	TextDecl ::= '<' '?' 'xml' VersionInfo? EncodingDecl S? '?>'
XMLDecl	public static final int XMLDecl	XMLDecl ::= '<' '?' 'xml' VersionInfo EncodingDecl? SDDecl? S? '?' '>'

token()

Receives an XML token and it's corresponding value. This is an interface call-back method.

Syntax

```
public void token( int token,
                   String value);
```

Parameter	Description
token	The XML token constant as specified in the interface.
value	The corresponding substring from the parsed text.

XMLTokenizer Class

This class implements an eXtensible Markup Language (XML) 1.0 parser according to the World Wide Web Consortium (W3C) recommendation.

Syntax

```
public class XMLTokenizer
   oracle.xml.parser.v2.XMLTokenizer
```

Table 1–15 Summary of Methods of XMLTokenizer

Method	Description
XMLTokenizer() on page 1-65	Creates a new Tokenizer object.
parseDocument() on page 1-66	Parses the document.
setErrorHandler() on page 1-66	Registers a new error event handler.
setErrorStream() on page 1-66	Registers a output stream for errors.
setToken() on page 1-66	Registers a new token for XML tokenizer.
setTokenHandler() on page 1-67	Registers a new XML tokenizer event handler.
tokenize() on page 1-67	Tokenizes the XML.

XMLTokenizer()

Creates a new Tokenizer object. The options are described in the following table.

Syntax	Description
`public XMLTokenizer();`	Creates a new Tokenizer object.
`public XMLTokenizer( XMLToken handler);`	Creates a new Tokenizer object from a handler.

Parameter	Description
`handler`	The handler.

parseDocument()

Parses the document.

Syntax

```
public void parseDocument();
```

setErrorHandler()

Registers a new error event handler. Replaces previous error handling settings.

Syntax

```
public void setErrorHandler(org.xml.sax.ErrorHandler handler)
```

Parameter	Description
handler	The handler being registered.

setErrorStream()

Registers a output stream for errors.

Syntax

```
public void setErrorStream( java.io.OutputStream out);
```

Parameter	Description
out	Error stream being registered.

setToken()

Registers a new token for XML tokenizer.

Syntax

```
public void setToken( int token,
                      boolean val);
```

Parameter	Description
token	XMLToken being set.

setTokenHandler()

Registers a new XML tokenizer event handler.

Syntax

```
public void setTokenHandler( XMLToken handler);
```

Parameter	Description
handler	XMLToken being registered.

tokenize()

Tokenizes the XML. The options are described in the following table. Throws the following exceptions:

- XMLParseException if syntax or other error encountered.
- SAXException, which could be any SAX exception, possibly wrapping another exception.
- IOException in cases of I/O Error.

Syntax	Description
public final void tokenize(org.xml.sax.InputSource in);	Tokenizes the XML from given input source.
public final void tokenize(java.io.InputStream in);	Tokenizes the XML from given input stream.
public final void tokenize(java.io.Reader r);	Tokenizes the XML from given reader.
public final void tokenize(String in);	Tokenizes the XML from a string.
public final void tokenize(java.net.URL url);	Tokenizes the XML document pointed to by the given URL and creates the corresponding XML document hierarchy.

Parameter	Description
in	The source for parsing.
url	The URL which points to the XML document to parse.

NSName Class

The NSName interface is part of the orace.xml.util package; it provides Namespace support for Element and Attr names.

Syntax

```
public interface oracle.xml.util.NSName
```

Table 1–16 Summary of Methods of NSName

Method	Description
getExpandedName() on page 1-69	Returns the fully resolved name for this name.
getLocalName() on page 1-69	Returns the local name for this name.
getNamespace() on page 1-69	Returns the resolved Namespace for this name.
getPrefix() on page 1-69	Returns the prefix for this name.
getQualifiedName() on page 1-70	Returns the qualified name.

getExpandedName()

Returns the fully resolved name for this name.

Syntax

```
public  String  getExpandedName();
```

getLocalName()

Returns the local name for this name.

Syntax

```
public String getLocalName();
```

getNamespace()

Returns the resolved Namespace for this name.

Syntax

```
public String getNamespace()
```

getPrefix()

Returns the prefix for this name.

Syntax

```
public String getPrefix();
```

getQualifiedName()

Returns the qualified name.

Syntax

```
public String getQualifiedName();
```

XMLError Class

This class of the `oracle.xml.util` package holds the error message and the line number where it occured.

Syntax

```
public class XMLError
   oracle.xml.util.XMLError
```

Table 1–17 Fields of oracle.xml.util.XMLError

Field	Syntax
col	protected int[] col
errid	protected int[] errid
exp	protected java.lang.Exception[] exp
line	protected int[] line
mesg	protected java.lang.String[] mesg
pubId	protected java.lang.String[] pubId
sysId	protected java.lang.String[] sysId
types	protected int[] types

Table 1–18 Summary of Methods of XMLError

Method	Description
XMLError() on page 1-73	Default constructor for XMLError.
error() on page 1-73	Adds a new error to the vector.
error0() on page 1-74	Adds a new error to the vector, with no additional parameters.
error1() on page 1-74	Adds a new error to the vector, with 1 additional parameter.
error2() on page 1-74	Adds a new error to the vector, with 2 additional parameters.
error3() on page 1-75	Adds a new error to the vector, with 3 additional parameters.

Table 1–18 (Cont.) Summary of Methods of XMLError

Method	Description
flushErrorStream() on page 1-75	Flush all the error to the output stream output stream defaults or to error handler
formatErrorMesg() on page 1-75	Formats and error message.
getColumnNumber() on page 1-76	Returns the column number of error at specified index
getException() on page 1-76	Returns the exception (if exists) that occured in error at specified index
getFirstError() on page 1-76	Returns the first error.
getLineNumber() on page 1-76	Returns the line number of error at specified index.
getLocator() on page 1-77	Returns the registered locator.
getMessage() on page 1-77	Returns the error message at specified index
getMessage0() on page 1-77	Returns error message with no parameters.
getMessage1() on page 1-78	Returns error message with 1 parameter.
getMessage2() on page 1-78	Returns error message with 2 parameters.
getMessage3() on page 1-78	Returns error message with 3 parameters.
getMessage4() on page 1-79	Returns error message with 4 parameters.
getMessage5() on page 1-80	Returns error message with 5 parameters.
getMessageType() on page 1-80	Returns the type of the error message at specified index.
getNumMessages() on page 1-80	Returns the total number of errors/warnings found during parsing.
getPublicId() on page 1-81	Returns the public ID of input when error at specified index occured.
getSystemId() on page 1-81	Returns the system ID of input when error at specified index occured.
printErrorListener() on page 1-81	Flushes all the JAXP 1.1 errors to the ErrorListener If no ErrorListener was set, default to System.err.
reset() on page 1-81	Resets the error class.
setErrorStream() on page 1-82	Registers an output stream.

Table 1–18 (Cont.) Summary of Methods of XMLError

Method	Description
setException() on page 1-82	Registers an exception.
setLocale() on page 1-82	Registers a locale.
setLocator() on page 1-83	Registers a locator.
showWarnings() on page 1-83	Turns reporting warning on/off.

XMLError()

Default constructor for XMLError.

Syntax

```
public  XMLError();
```

error()

Adds a new error to the vector. The options are described in the following table.

Syntax	Description
`public void error( int id, int type, String msg);`	Adds a new error to the vector, with a message.
`public void error3( int id, int type, String[] params);`	Adds a new error to the vector, with an array of parameters.

Parameter	Description
`id`	Id of the error message.
`type`	Type of the error.
`MESG`	Error message (without parameters).
`params`	Parameter array.

error0()

Adds a new error to the vector, with no parameters.

Syntax

```
public void error3( int id,
                    int type);
```

Parameter	Description
id	Id of the error message.
type	Type of the error.

error1()

Adds a new error to the vector, with 1 parameter.

Syntax

```
public void error3( int id,
                    int type,
                    String p1);
```

Parameter	Description
id	Id of the error message.
type	Type of the error.
p1	Parameter 1.

error2()

Adds a new error to the vector, with two parameters.

Syntax

```
public void error3( int id,
                    int type,
                    String p1,
                    String p2);
```

Parameter	Description
id	Id of the error message.
type	Type of the error.
p1	Parameter 1.
p2	Parameter 2.

error3()

Adds a new error to the vector, with 3 parameters.

Syntax

```
public void error3( int id,
                    int type,
                    String p1,
                    String p2,
                    String p3)
```

Parameter	Description
id	Id of the error message.
type	Type of the error.
p1	Parameter 1.
p2	Parameter 2.
p3	Parameter 3.

flushErrorStream()

Flushes all the error to the output stream output stream defaults or to error handler.

Syntax

```
public void flushErrorStream();
```

formatErrorMesg()

Formats an error message.

Syntax

```
public String formatErrorMesg(int index);
```

Parameter	Description
i	Index.

getColumnNumber()

Returns the column number of error at specified index.

Syntax

```
public int getColumnNumber(int i);
```

Parameter	Description
i	Index.

getException()

Returns the exception (if exists) that occured in error at specified index.

Syntax

```
public java.lang.Exception getException(int i);
```

Parameter	Description
i	Index.

getFirstError()

Returns the first error.

Syntax

```
public int getFirstError();
```

getLineNumber()

Returns the line number of error at specified index.

Syntax

```
public int getLineNumber(int i);
```

Parameter	Description
i	Index.

getLocator()

Returns the registered locator

Syntax

```
public org.xml.sax.Locator getLocator();
```

getMessage()

Returns an error message. The options are described in the following table.

Syntax	Description
`public String getMessage( int i);`	Returns the error message at specified index.
`public String getMessage( int errId, String[] params);`	Returns error message with more than 5 parameters, packaged in an array.

Parameter	Description
i	Index.
errId	The error id.
param	An array of parameters.

getMessage0()

Returns error message with no parameters.

Syntax

```
public String getMessage1( int errId);
```

Parameter	Description
errId	The error id.

getMessage1()

Returns error message with 1 argument.

Syntax

```
public String getMessage1( int errId,
                           String a1);
```

Parameter	Description
errId	The error id.
a1	Parameter 1.

getMessage2()

Returns error message with 2 parameters

Syntax

```
public String getMessage3(
     int errId,
     String a1,
     String a2);
```

Parameter	Description
errId	The error id.
a1	Parameter 1.
a2	Parameter 2.

getMessage3()

Returns error message with 3 parameters.

Syntax

```
public String getMessage3(
     int errId,
     String a1,
     String a2,
     String a3);
```

Parameter	Description
errId	The error id.
a1	Parameter 1.
a2	Parameter 2.
a3	Parameter 3.

getMessage4()

Returns error message with 4 parameters.

Syntax

```
public String getMessage4(
     int errId,
     String a1,
     String a2,
     String a3,
     String a4);
```

Parameter	Description
errId	The error id.
a1	Parameter 1.
a2	Parameter 2.
a3	Parameter 3.
a4	Parameter 4.

getMessage5()

Returns error message with 5 parameters.

Syntax

```
public String getMessage5(
     int errId,
     String a1,
     String a2,
     String a3,
     String a4,
     String a5);
```

Parameter	Description
errId	The error id.
a1	Parameter 1.
a2	Parameter 2.
a3	Parameter 3.
a4	Parameter 4.
a5	Parameter 5.

getMessageType()

Returns the type of the error message type at specified index

Syntax

```
public int getMessageType(int i);
```

Parameter	Description
i	Index

getNumMessages()

Returns the total number of errors/warnings found during parsing

Syntax

```
public int getNumMessages()
```

getPublicId()

Returns the public ID of input when error at specified index occured

Syntax

```
public String getPublicId( int i);
```

Parameter	Description
i	Index.

getSystemId()

Returns the system ID of input when error at specified index occured.

Syntax

```
public String getSystemId(int i);
```

Parameter	Description
i	Index.

printErrorListener()

Flushes all the JAXP 1.1 errors to the ErrorListener. If no ErrorListener was set, defaults to System.err.

Syntax

```
public void printErrorListener();
```

reset()

Resets the error class.

Syntax

```
public void reset();
```

setErrorStream()

Sets an output stream for error reporting. The options are described in the following table.

Syntax	Description
`public void setErrorStream( java.io.OutputStream out);`	Sets an output stream for error reporting. Throws `IOException` if an error occurs initializing the output stream.
`public void setErrorStream( java.io.OutputStream out, String enc);`	Sets an output stream for error reporting and its encoding. Throws `IOException` if an error occurs initializing the output stream.
`public void setErrorStream( java.io.PrintWriter out);`	Sets an print writer for error reporting.

Parameter	Description
`out`	Output for errors/warnings.
`enc`	Encoding of the output stream.

setException()

Registers an exception.

Syntax

```
public void setException( java.lang.Exception exp);
```

Parameter	Description
`exp`	Last exception which occured.

setLocale()

Registers a locale for error reporting.

Syntax

```
public void setLocale( java.util.Locale locale);
```

Parameter	Description
locale	The locale for error reporting.

setLocator()

Register a locator

Syntax

```
public void setLocator( org.xml.sax.Locator locator);
```

Parameter	Description
locator	Locator to get lin/col/sysid/pubid information.

showWarnings()

Turns reporting warning on/off.

Syntax

```
public void showWarnings(boolean flag);
```

Parameter	Description
flag	Controls reporting of warnings.

XMLException Class

The XML Exception class in package `oracle.xml.util` indicates that a parsing exception occurred while processing an XML document.

Syntax

```
public class XMLException extends java.lang.Exception
```

Table 1–19 Fields of XMLException

Field	Syntax	Description
`ERROR`	`public static final int ERROR`	Code for non-fatal error
`FATAL_ERRORS`	`public static final int FATAL_ERRORS`	Code for fatal error
`WARNING`	`public static final int WARNINGS`	Code for warning

Table 1–20 Summary of Methods of oracle.xml.util.XMLException

Method	Description
XMLException() on page 1-85	Generates an XMLException.
formatErrorMessage() on page 1-86	Returns the error message at specified index.
getColumnNumber() on page 1-86	Returns the column number of error at specified index.
getException() on page 1-86	Returns the exception (if exists) that occured in error at specified index.
getLineNumber() on page 1-87	Returns the line number of error at specified index.
getMessage() on page 1-87	Returns the error message at specified index.
getMessageType() on page 1-87	Returns the type of the error message at specified index.
getNumMessages() on page 1-88	Returns the total number of errors/warnings found during parsing.
getPublicId() on page 1-88	Returns the public ID of input when error at specified index occured.

Table 1–20 (Cont.) Summary of Methods of oracle.xml.util.XMLException

Method	Description
getSystemId() on page 1-88	Returns the system ID of input when error at specified index occured.
getXMLError() on page 1-88	Returns XMLError object inside XMLException.
printStackTrace() on page 1-89	Prints this `Throwable` and its backtrace.
setException() on page 1-89	Sets the underlying exception, if exists.
toString() on page 1-89	Returns any embedded exception.

XMLException()

Generates an XML Exception. The options are described in the following table.

Syntax	Description
`public XMLException( String mesg, String pubId, String sysId, int line, int col, int type);`	Generates and XMLException from the message, public id, system id, line, column and type.
`public XMLException( XMLError err, java.lang.Exception e);`	Generates and XMLException from the error and exception.
`public XMLException( XMLError err, int firsterr);`	Generates and XMLException from the error and first error.
`public XMLException( XMLError err, int firsterr, java.lang.Exception e);`	Generates and XMLException from the error, exception and first error.

Parameter	Description
`mesg`	Message.
`pubId`	Public Id.

Parameter	Description
sysId	System Id.
line	Line.
col	Column.
type	Type.
err	Error.
e	Exception.
firsterr	First error.

formatErrorMessage()

Returns the error message at specified index.

Syntax

```
public String formatErrorMessage( int i);
```

Parameter	Description
i	Index.

getColumnNumber()

Returns the column number of error at specified index.

Syntax

```
public int getColumnNumber( int i);
```

Parameter	Description
i	Index.

getException()

Returns the exception (if exists) that occured in error at specified index

Syntax

```
public java.lang.Exception getException( int i);
```

Parameter	Description
i	Index

getLineNumber()

Returns the line number of error at specified index

Syntax

```
public int getLineNumber(int i);
```

Parameter	Description
i	Index.

getMessage()

Returns the error message at specified index

Syntax

```
public String getMessage(int i).
```

Parameter	Description
i	Index.

getMessageType()

Get the type of the error message at specified index

Syntax

```
public int getMessageType(int i);
```

Parameter	Description
i	Index.

getNumMessages()

Returns the total number of errors/warnings found during parsing

Syntax

```
public int getNumMessages();
```

getPublicId()

Returns the public ID of input when error at specified index occured

Syntax

```
public String getPublicId(int i);
```

Parameter	Description
i	Index.

getSystemId()

Returns the system ID of input when error at specified index occured

Syntax

```
public String getSystemId(int i);
```

Parameter	Description
i	Index.

getXMLError()

Get XMLError object inside XMLException.

Syntax

```
public XMLError getXMLError();
```

printStackTrace()

Prints `Throwable` and its backtrace. The options are described in the following table.

Syntax	Description
`public void printStackTrace();`	Prints this `Throwable` and its backtrace to the standard error stream.
`public void printStackTrace( java.io.PrintStream s);`	Prints this Throwable and its backtrace to the specified print stream.
`public void printStackTrace( java.io.PrintWriter s);`	Prints this Throwable and its backtrace to the specified print writer.

Parameter	Description
`s`	Output for the stack trace.

setException()

Sets the underlying exception, if it exists.

Syntax

```
public void setException(java.lang.Exception ex);
```

Parameter	Description
`ex`	The exception

toString()

Returns any embedded exception.

Syntax

```
public String toString();
```

2

Document Object Model (DOM)

An XML document can be viewed as a tree whose nodes consisting of information between start-tags and end-tags. This tree representation, or the Document Object Model (DOM), is formed in memory when the XML parser parses a document. The DOM APIs are contained in the oracle.xml.parser.v2 package.

This chapter discusses the APIs for accessing and navigating this tree and includes the following sections:

- NSResolver Interface
- PrintDriver Interface
- AttrDecl Class
- DTD Class
- ElementDecl Class
- XMLAttr Class
- XMLCDATA Class
- XMLComment Class
- XMLDeclPI Class
- XMLDocument Class
- XMLDocumentFragment Class
- XMLDOMException Class
- XMLDOMImplementation
- XMLElement Class
- XMLEntity Class

- XMLEntityReference Class
- XMLNode Class
- XMLNotation Class
- XMLNSNode Class
- XMLOutputStream Class
- XMLPI Class
- XMLPrintDriver Class
- XMLRangeException Class
- XMLText Class

See Also:

- *Oracle Application Developer's Guide - XML*

NSResolver Interface

This interface provides support for resolving Namespaces.

Syntax

```
public interface NSResolver
```

resolveNamespacePrefix()

Finds and returns the namespace definition in scope for a given namespace prefix; returns null if prefix could not be resolved.

Syntax

```
public String resolveNamespacePrefix( String prefix);
```

Parameter	Description
`prefix`	Namespace prefix to be resolved.

PrintDriver Interface

The `PrintDriver` interface defines methods used to print XML documents represented as DOM trees.

Syntax

```
public interface PrintDriver
```

Table 2–1 Summary of Methods of PintDriver

Method	Description
close() on page 2-5	Closes the output stream or print writer
flush() on page 2-5	Flushes the output stream or print writer
printAttribute() on page 2-5	Prints an XMLAttr node
printAttributeNodes() on page 2-5	Calls print method for each attribute of the XMLElement.
printCDATASection() on page 2-5	Prints an XMLCDATA node.
printChildNodes() on page 2-5	Calls print method for each child of the XMLNode
printComment() on page 2-6	Prints an XMLComment node.
printDoctype() on page 2-6	Prints a DTD.
printDocument() on page 2-7	Prints an XMLDocument.
printDocumentFragment() on page 2-7	Prints an empty XMLDocumentFragment object.
printElement() on page 2-7	Prints an XMLElement.
printEntityReference() on page 2-7	Prints an XMLEntityReference node.
printProcessingInstruction() on page 2-8	Prints an XMLPI node.
printTextNode() on page 2-8	Prints an XMLText node.
setEncoding() on page 2-8	Sets the encoding of the print driver.

close()

Closes the output stream or print writer

Syntax

```
public void close();
```

flush()

Flushes the output stream or print writer

Syntax

```
public void flush();
```

printAttribute()

Prints an `XMLAttr` node

Syntax

```
public void printAttribute( XMLAttr attr);
```

Parameter	Description
`attr`	The XMLAttr node.

printAttributeNodes()

Calls print method for each attribute of the `XMLElement`.

Syntax

```
public void printAttributeNodes( XMLElement elem);
```

Parameter	Description
`elem`	The elem whose attributes are to be printed.

printCDATASection()

Prints an `XMLCDATA` node.

Syntax

```
public void printCDATASection( XMLCDATA cdata);
```

Parameter	Description
cdata	The XMLCDATA node.

printChildNodes()

Calls print method for each child of the XMLNode

Syntax

```
public void printChildNodes( XMLNode node);
```

Parameter	Description
node	The node whose children are to be printed.

printComment()

Prints an XMLComment node.

Syntax

```
public void printComment( XMLComment comment);
```

Parameter	Description
comment	The comment node.

printDoctype()

Prints a DTD.

Syntax

```
public void printDoctype( DTD dtd);
```

Parameter	Description
dtd	The DTD to be printed.

printDocument()

Prints an `XMLDocument`.

Syntax

```
public void printDocument( XMLDocument doc);
```

Parameter	Description
`doc`	The document to be printed.

printDocumentFragment()

Prints an empty `XMLDocumentFragment` object.

Syntax

```
public void printDocumentFragment( XMLDocumentFragment dfrag);
```

Parameter	Description
`dfrag`	The document fragment to be printed.

printElement()

Prints an `XMLElement`.

Syntax

```
public void printElement( XMLElement elem);
```

Parameter	Description
`elem`	The element to be printed.

printEntityReference()

Prints an `XMLEntityReference` node.

Syntax

```
public void printEntityReference( XMLEntityReference eref);
```

Parameter	Description
eref	The XMLEntityReference node to be printed.

printProcessingInstruction()

Prints an XMLPI node.

Syntax

```
public void printProcessingInstruction( XMLPI pi);
```

Parameter	Description
pi	The XMLPI node to be printed.

printTextNode()

Prints an XMLText node.

Syntax

```
public void printTextNode( XMLText text);
```

Parameter	Description
text	The text node.

setEncoding()

Sets the encoding of the print driver.

Syntax

```
public void setEncoding( String enc);
```

Parameter	Description
enc	The encoding of the document being printed.

AttrDecl Class

This class hold information about each attribute declared in an attribute list in the Document Type Definition.

Syntax

```
public class AttrDecl implements java.io.Externalizable
```

Table 2–2 Fields of AttrDecl

Field	Syntax	Description
CDATA	public static final int CDATA	AttType - StringType - CDATA
DEFAULT	public static final int DEFAULT	Attribute presence - Default
ENTITIES	public static final int ENTITIES	AttType - TokenizedType - Entities
ENTITY	public static final int ENTITY	AttType - TokenizedType - Entity
ENUMERATION	public static final int ENUMERATION	AttType - EnumeratedType - Enumeration
FIXED	public static final int FIXED	Attribute presence - Fixed
ID	public static final int ID	AttType - TokenizedType - ID
IDREF	public static final int IDREF	AttType - TokenizedType - ID reference
IDREFS	public static final int IDREFS	AttType - TokenizedType - ID references
IMPLIED	public static final int IMPLIED	Attribute presence - Implied
NMTOKEN	public static final int NMTOKEN	AttType - TokenizedType - Name token
NMTOKENS	public static final int NMTOKENS	AttType - TokenizedType - Name tokens

Table 2–2 (Cont.) Fields of AttrDecl

Field	Syntax	Description
NOTATION	public static final int NOTATION	AttType - EnumeratedType - Notation
REQUIRED	public static final int REQUIRED	Attribute presence - Required

Table 2–3 Summary of Methods of AttrDecl

Methods	Description
AttrDecl() on page 2-11	Default constructor.
getAttrPresence() on page 2-12	Returns attribute presence.
getAttrType() on page 2-12	Returns attribute type.
getDefaultValue() on page 2-12	Returns attribute default value.
getEnumerationValues() on page 2-12	Returns attribute values as an Enumeration.
getNodeName() on page 2-12	Returns the name of the Attr Decl.
getNodeType() on page 2-12	Returns a code representing the type of the underlying object.
readExternal() on page 2-12	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
typeToString() on page 2-13	Returns a string representation of the attribute type.
writeExternal() on page 2-13	Saves the state of the object by creating a binary compressed stream with information about this object.

AttrDecl()

Default constructor.

Syntax

```
public static final int REQUIRED public  AttrDecl();
```

getAttrPresence()

Returns attribute presence.

Syntax

```
public int getAttrPresence();
```

getAttrType()

Returns attribute type.

Syntax

```
public int getAttrType();
```

getDefaultValue()

Returns attribute default value.

Syntax

```
public String getDefaultValue();
```

getEnumerationValues()

Returns attribute values as an `Enumeration`.

Syntax

```
public java.util.Vector getEnumerationValues();
```

getNodeName()

Returns the name of the Attr Decl.

Syntax

```
public String getNodeName();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

readExternal()

Restores the object after reading it from a compressed stream. Throws:

- `IOException` is thrown when there is an error in reading the input stream.

- `ClassNotFoundException` is thrown when the class is not found

Syntax

```
public void readExternal( java.io.ObjectInput i);
```

Parameter	Description
`in`	The ObjectInput stream used for reading the compressed stream.

typeToString()

Returns a string representation of the attribute type.

Syntax

```
public static String typeToString( int type);
```

Parameter	Description
`type`	Numerical representation of the attribute type.

writeExternal()

Saves the state of the object by creating a binary compressed stream. Throws `IOException` if serialized/compressed stream.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
`out`	The ObjectOutput stream used to write the stream.

DTD Class

Implements the DOM DocumentType interface and holds the Document Type. Definition information for an XML document.

Syntax

```
public class DTD implements java.io.Externalizable
```

Table 2–4 Summary of Methods of DTD

Method	Description
DTD() on page 2-15	Default constructor.
findElementDecl() on page 2-15	Finds and returns an element declaration for the given tag name.
findEntity() on page 2-16	Finds and returns a named entity in the DTD; returns null if it is not found.
findNotation() on page 2-16	Retrieves the named notation from the DTD; returns null if it is not found.
getChildNodes() on page 2-16	Returns a NodeList that contains all children of this node.
getElementDecls() on page 2-16	Returns a NamedNodeMap containing the element declarations in the DTD.
getEntities() on page 2-17	Returns a NamedNodeMap containing the general entities, both external and internal, declared in the DTD.
getInternalSubset() on page 2-17	Returns the internal subset of the DTD.
getName() on page 2-17	Returns the name of the DTD, or the name immediately following the DOCTYPE keyword.
getNodeName() on page 2-17	Returns the name of the DTD, or the name immediately following the DOCTYPE keyword.
getNodeType() on page 2-17	Returns a code representing the type of the underlying object.
getNotations() on page 2-18	Returns a NamedNodeMap containing the notations declared in the DTD.
getOwnerImplementation( ) on page 2-18	Returns the owner of the DTD implementation.

Table 2–4 (Cont.) Summary of Methods of DTD

Method	Description
getPublicId() on page 2-18	Returns the public identifier associated with the DTD, if specified.
getRootTag() on page 2-18	Returns the root tag for the DTD.
getSystemId() on page 2-18	Returns the system identifier associated with the DTD, if specified. If the system identifier was not specified, this is null.
hasChildNodes() on page 2-18	Determines whether a node has any children; return false always, as DTD cannot have any overrides method in XMLNode.
normalize() on page 2-19	Normalizes the DTD.
printExternalDTD() on page 2-19	Writes the contents of this document to the given output.
readExternal() on page 2-19	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
setRootTag() on page 2-20	Sets the root tag for the DTD.
writeExternal() on page 2-20	Saves the state of the object by creating a binary compressed stream with information about this object.

DTD()

Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node. In order to deserialize this node to construct the DOM node from the serialized/ compressed stream, it is required to create a handle of the object.

Syntax

```
public  DTD();
```

findElementDecl()

Finds and returns an element declaration for the given tag name.

Syntax

```
public final ElementDecl findElementDecl( String name);
```

Parameter	Description
name	The tag name.

findEntity()

Finds and returns a named entity in the DTD; returns null if it is not found.

Syntax

```
public final org.w3c.dom.Entity findEntity( String n,
                                            boolean par);
```

Parameter	Description
n	The name of the entity.
par	Boolean indicating if the entity is parameter Entity.

findNotation()

Retrieves the named notation from the DTD; returns null if it is not found.

Syntax

```
public final org.w3c.dom.Notation findNotation( String name);
```

Parameter	Description
name	The name of the notation.

getChildNodes()

Return a `NodeList` that contains all children of this node. If there are no children, this is a `NodeList` containing no nodes. The content of the returned `NodeList` is "live" in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the `NodeList` accessors; it is not a static snapshot of the content of the node. This is true for every `NodeList`, including the ones returned by the `getElementsByTagName` method.

Syntax

```
public org.w3c.dom.NodeList getChildNodes();
```

getElementDecls()

Returns a `NamedNodeMap` containing the element declarations in the DTD. Every node in this map is an `ElementDecl` object. The element declarations in the DTD

Syntax

```
public org.w3c.dom.NamedNodeMap getElementDecls();
```

getEntities()

Returns a `NamedNodeMap` containing the general entities, both external and internal, declared in the DTD. Duplicates are discarded. Every node in this map also implements the `Entity` interface.

Syntax

```
public org.w3c.dom.NamedNodeMap getEntities();
```

getInternalSubset()

Returns the internal subset of the DTD.

Syntax

```
public String getInternalSubset();
```

getName()

Returns the name of the DTD, or the name immediately following the `DOCTYPE` keyword.

Syntax

```
public String getName();
```

getNodeName()

Returns the name of the DTD; or the name immediately following the `DOCTYPE` keyword.

Syntax

```
public String getNodeName();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

getNotations()

Returns a `NamedNodeMap` containing the notations declared in the DTD. Duplicates are discarded. Every node in this map also implements the `Notation` interface. The DOM Level 1 does not support editing notations, therefore `notations` cannot be altered in any way.

Syntax

```
public org.w3c.dom.NamedNodeMap getNotations();
```

getOwnerImplementation()

Returns the owner of the DTD implementation.

Syntax

```
public XMLDOMImplementation getOwnerImplementation();
```

getPublicId()

Returns the public identifier associated with the DTD, if specified. If the public identifier was not specified, this is `null`.

Syntax

```
public String getPublicId();
```

getRootTag()

Returns the root tag for the DTD.

Syntax

```
public String getRootTag();
```

getSystemId()

Returns the system identifier associated with the DTD, if specified. If the system identifier was not specified, this is `null`.

Syntax

```
public String getSystemId();
```

hasChildNodes()

Determines whether a node has any children; return false always, as DTD cannot have any overrides method in XMLNode.

Syntax

```
public boolean hasChildNodes();
```

normalize()

Normalizes the DTD.

Syntax

```
public void normalize();
```

printExternalDTD()

Writes the contents of this document to the given output. Throws `IOException` if an invalid encoding is specified or another error occurs.

Syntax	Description
`public void printExternalDTD( java.io.OutputStream out);`	Writes the contents of this document to the given output stream.
`public void printExternalDTD( java.io.OutputStream out, String enc);`	Writes the contents of this document to the given encoded output stream.
`public void printExternalDTD( java.io.PrintWriter out);`	Writes the contents of this document to the given print writer.

Parameter	Description
`out`	The output.
`enc`	Encoding used for the output.

readExternal()

Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:

- `IOException` when there is an error in reading the input stream.
- `ClassNotFoundException` when the class is not found.

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
in	The ObjectInput stream used for reading the compressed stream.

setRootTag()

Set the root tag for the DTD

Syntax

```
public void setRootTag( String root);
```

Parameter	Description
root	The root tag.

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws `IOException` when there is an exception while writing the serialized/compressed stream.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
out	The ObjectOutput stream

ElementDecl Class

This class represents an element declaration in a DTD.

Syntax

```
public class ElementDecl implements java.io.Serializable, java.io.Externalizable
```

Table 2–5 Fields of ElementDecl

Field	Syntax	Description
ANY	public static final byte ANY	Element content type - Children can be any element.
ASTERISK	public static final int ASTERISK	ContentModelParseTreeNode type - "*" node (has one child).
COMMA	public static final int COMMA	ContentModelParseTreeNode type - "," node (has two children).
ELEMENT	public static final int ELEMENT	ContentModelParseTreeNode type - 'leaf' node (has no children).
ELEMENT_DECLARED	public static final int ELEMENT_DECLARED	Node flag to represent element declaration in a DTD.
ELEMENTS	public static final byte ELEMENTS	Element content type - Children can be elements; see Content Model.
EMPTY	public static final byte EMPTY	Element content type - No Children.
ID_ATTR_DECL	public static final int ID_ATTR_DECL	Node flag to represent ID attribute declaration in a DTD.
MIXED	public static final byte MIXED	Element content type - Children can be PCDATA & elements; see Content Model.
OR	public static final int OR	ContentModelParseTreeNode type - "|" node (has two children).

Table 2–5 (Cont.) Fields of ElementDecl

Field	Syntax	Description
PLUS	public static final int PLUS	ContentModelParseTreeNode type - "+" node (has one children).
QMARK	public static final int QMARK	ContentModelParseTreeNode type - "?" node (has one children).

Table 2–6 Summary of Methods of ElementDecl

Method	Description
ElementDecl() on page 2-23	Default constructor.
cloneNode() on page 2-23	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
expectedElements() on page 2-23	Returns vector of element names that can be appended to the element.
findAttrDecl() on page 2-24	Returns an attribute declaration object or null if not found
getAttrDecls() on page 2-24	Returns an enumeration of attribute declarations.
getContentElements() on page 2-24	Returns a vector of elements that can be appended to this element.
getContentType() on page 2-24	Returns the content model of element.
getNodeName() on page 2-24	Returns the name of the Element Decl.
getNodeType() on page 2-24	Returns a code representing the type of the underlying object.
getParseTree() on page 2-25	Returns the root node of Content Model Parse Tree.
readExternal() on page 2-25	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:
validateContent() on page 2-25	Validates the content of a element node.
writeExternal() on page 2-25	Saves the state of the object by creating a binary compressed stream with information about this object.

ElementDecl()

Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node. In order to deserialize this node to construct the DOM node from the serialized/ compressed stream, it is required to create a handle of the object.

Syntax

```
public  ElementDecl();
```

cloneNode()

Returns a duplicate of this node; serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns `null`.). Cloning an `Element` copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child `Text` node. Cloning any other type of node simply returns a copy of this node.

Syntax

```
public org.w3c.dom.Node cloneNode(boolean deep);
```

Parameter	Description
`deep`	If `true`, recursively clone the subtree under the specified node; if `false`, clone only the node itself (and its attributes, if it is an `Element`)

expectedElements()

Returns vector of element names that can be appended to the element.

Syntax

```
public java.util.Vector expectedElements( org.w3c.dom.Element e);
```

Parameter	Description
`e`	Element.

findAttrDecl()

Returns an attribute declaration object or null if not found

Syntax

```
public final AttrDecl findAttrDecl( String name);
```

Parameter	Description
name	The attribute declaration to find.

getAttrDecls()

Returns an enumeration of attribute declarations.

Syntax

```
public org.w3c.dom.NamedNodeMap getAttrDecls();
```

getContentElements()

Returns a vector of elements that can be appended to this element.

Syntax

```
public final java.util.Vector getContentElements();
```

getContentType()

Returns the content model of element.

Syntax

```
public int getContentType();
```

getNodeName()

Returs the name of the Element Decl.

Syntax

```
public String getNodeName();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

getParseTree()

Returns the root node of Content Model Parse Tree. `Node.getFirstChild()` and `Node.getLastChild()` return the parse tree branches. `Node.getNodeType()` and `Node.getNodeName()` return the parse tree node type and name.

Syntax

```
public final org.w3c.dom.Node getParseTree();
```

readExternal()

Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:

- `IOException` when there is an error in reading the input stream.
- `ClassNotFoundException` when the class is not found.

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
`in`	The ObjectInput stream used for reading the compressed stream.

validateContent()

Validates the content of a element node; returns `TRUE` if valid, `FLASE` otherwise.

Syntax

```
public boolean validateContent(org.w3c.dom.Element e);
```

Parameter	Description
`e`	Element node to validate.

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws `IOException` when there is an exception while writing the serialized/compressed stream.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
out	The ObjectOutput stream used to write the serialized/ compressed stream.

XMLAttr Class

This class implements the DOM Attr interface and holds information on each attribute of an element. See also `Attr, NodeFactory, DOMParser.setNodeFactory()`.

Syntax

```
public class XMLAttr implements oracle.xml.parser.v2.NSName,
java.io.Externalizable
```

Table 2–7 Summary of Methods of XMLAttr

Method	Description
addText() on page 2-28	Adds text to the XMLNode.
cloneNode() on page 2-28	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
getExpandedName() on page 2-29	Returns the fully resolved Name for this attribute.
getLocalName() on page 2-29	Returns the local name of this attribute.
getName() on page 2-29	Returns the attribute name.
getNamespaceURI() on page 2-29	Returns the namespace of the attribute.
getNextAttribute() on page 2-29	Returns the next attribute, if any.
getNextSibling() on page 2-29	Returns the next sibling, if any.
getNodeType() on page 2-29	Returns a code representing the type of the underlying object.
getNodeValue() on page 2-30	Returns the value of this node, depending on its type.
getOwnerElement() on page 2-30	Returns the element which owns this attribute.
getParentNode() on page 2-30	Returns the parent of this node.
getPrefix() on page 2-30	Returns the name space prefix of the element.
getPreviousSibling() on page 2-30	Returns the previous sibling, if any.
getSpecified() on page 2-30	Returns TRUE if the attribute was specified explicitly in the element, FALSE otherwise.
getValue() on page 2-31	Returns the attribute value.

Table 2–7 (Cont.) Summary of Methods of XMLAttr

Method	Description
readExternal() on page 2-31	Restores the information written by writeExternal.
setNodeValue() on page 2-31	Sets the value of this node, depending on its type.
setValue() on page 2-32	Sets the value.
writeExternal() on page 2-32	Saves the state of the object in a binary compressed stream.

addText()

Adds text to the XMLNode.

Syntax

```
public XMLNode addText( String str);
```

Parameter	Description
`str`	Text added.

cloneNode()

Returns a duplicate of this node; serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns `null`.). Cloning an `Element` copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child `Text` node. Cloning any other type of node simply returns a copy of this node.

Syntax

```
public org.w3c.dom.Node cloneNode( boolean deep);
```

Parameter	Description
`deep`	If `true`, recursively clone the subtree under the specified node; if `false`, clone only the node itself (and its attributes, if it is an `Element`).

getExpandedName()

Returns the fully resolved Name for this attribute.

Syntax

```
public String getExpandedName();
```

getLocalName()

Returns the local name of this attribute.

Syntax

```
public String getLocalName();
```

getName()

Returns the attribute name.

Syntax

```
public String getName();
```

getNamespaceURI()

Returns the namespace of the attribute.

Syntax

```
public String getNamespaceURI();
```

getNextAttribute()

Returns the next attribute, if any.

Syntax

```
public XMLAttr getNextAttribute();
```

getNextSibling()

Returns the next sibling, if any.

Syntax

```
public org.w3c.dom.Node getNextSibling();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

getNodeValue()

Returns the value of this node, depending on its type. Throws `DOMException`:

- `NO_MODIFICATION_ALLOWED_ERR` raised when the node is readonly.
- `DOMSTRING_SIZE_ERR` raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

Syntax

```
public String getNodeValue();
```

getOwnerElement()

Returns the element which owns this attribute.

Syntax

```
public org.w3c.dom.Element getOwnerElement();
```

getParentNode()

Returns the parent of this node. All nodes, except `Document`, `DocumentFragment`, and `Attr` may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is `null`.

Syntax

```
public org.w3c.dom.Node getParentNode();
```

getPrefix()

Returns the name space prefix of the element.

Syntax

```
public String getPrefix();
```

getPreviousSibling()

Returns the previous sibling, if any.

Syntax

```
public org.w3c.dom.Node getPreviousSibling();
```

getSpecified()

Returns `TRUE` if the attribute was specified explicitly in the element, `FALSE` otherwise.

Syntax

```
public boolean getSpecified();
```

getValue()

Returns the attribute value.

Syntax

```
public String getValue();
```

readExternal()

Restores the information written by writeExternal. Throws the following exceptions:

- `IOException` when there is an exception while reading the compressed stream.
- `ClassNotFoundException` when the class is not found

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
`in`	The ObjectInput stream used to read the compressed stream.

setNodeValue()

Sets the value of this node, depending on its type. Throws `DOMException`:

- `NO_MODIFICATION_ALLOWED_ERR` raised when the node is readonly.
- `DOMSTRING_SIZE_ERR` raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

Syntax

```
public void setNodeValue( String nodeValue);
```

Parameter	Description
`nodeValue`	The value of the node to be set.

setValue()

Sets the value.

Syntax

```
public void setValue( String val);
```

Parameter	Description
val	The value to set.

writeExternal()

Saves the state of the object in a binary compressed stream. Throws `IOException` when there is an exception while writing the compressed stream.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
out	The ObjectOutput stream used to write the compressed stream.

XMLCDATA Class

This class implements the DOM CDATASection interface. See also `CDATASection, NodeFactory, DOMParser.setNodeFactory(NodeFactory)`.

Syntax

```
public class XMLCDATA implements java.io.Externalizable
```

Table 2–8 Summary of Methods of XMLCDATA

Method	Description
XMLCDATA() on page 2-33	Default constructor.
getNodeName() on page 2-33	Returns the name of the node.
getNodeType() on page 2-34	Returns a code representing the type of the underlying object.
readExternal() on page 2-34	Reads the information written in the compressed stream by `writeExternal()` method and restores the object correspondingly.
writeExternal() on page 2-34	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLCDATA()

Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node. In order to deserialize this node to construct the DOM node from the serialized/ compressed stream, it is required to create a handle of the object.

Syntax

```
public  XMLCDATA();
```

getNodeName()

Returns the name of the node.

Syntax

```
public String getNodeName();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

readExternal()

Reads the information written in the compressed stream by `writeExternal()` method and restores the object correspondingly. Throws the following exceptions:

- `IOException` when there is an error in reading the input stream.
- `ClassNotFoundException` when the class is not found.

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
`in`	The ObjectInput stream used for reading the compressed stream.

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws `IOException` when there is an exception while writing the compressed stream.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
`out`	The ObjectOutput stream used to write the compressed stream.

XMLComment Class

This class implements the DOM Comment interface. See also `Comment, NodeFactory, DOMParser.setNodeFactory()`.

Syntax

```
public class XMLComment implements java.io.Externalizable
```

Table 2–9 Summary of XMLComment

Method	Description
XMLComment() on page 2-35	Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node.
addText() on page 2-35	Adds the comment text.
getNodeName() on page 2-36	Returns a name of the node.
getNodeType() on page 2-36	Returns a code representing the type of the underlying object.
readExternal() on page 2-36	Reads the information in the compressed stream and restores the object correspondingly.
reportSAXEvents() on page 2-36	Reports SAX Events from a DOM Tree.
writeExternal() on page 2-37	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLComment()

Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node. In order to deserialize this node to construct the DOM node from the serialized/ compressed stream, it is required to create a handle of the object.

Syntax

```
public  XMLComment();
```

addText()

Adds the comment text.

Syntax

```
public XMLNode addText( String str);
```

Parameter	Description
str	The comment text.

getNodeName()

Returns a name of the node.

Syntax

```
public String getNodeName();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

readExternal()

Reads the information written in the compressed stream by `writeExternal()` method and restores the object correspondingly. Throws the following exceptions:

- `IOException` when there is an error in reading the input stream.
- `ClassNotFoundException` when the class is not found

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
in	The ObjectInput stream used for reading the compressed stream.

reportSAXEvents()

Reports SAX Events from a DOM Tree. Throws `SAXException`.

Syntax

```
public void reportSAXEvents( org.xml.sax.ContentHandler cntHandler);
```

Parameter	Description
cntHandler	Content handler.

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws `IOException` when there is an exception while writing the compressed stream.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
out	The ObjectOutput stream used to write the compressed stream.

XMLDeclPI Class

This class implements the XML Decl Processing Instruction. See also XMLPI Class.

Syntax

```
public class XMLDeclPI extends oracle.xml.parser.v2.XMLPI implements
java.io.Externalizable
```

Table 2–10 Summary of Methods of XMLDeclPI

Method	Description
XMLDeclPI() on page 2-39	Creates a new instance of XMLDeclPI.
cloneNode() on page 2-39	Returns a duplicate of this node; serves as a generic copy constructor.
getData() on page 2-39	Returns the fully constructed string 'version=1.0....'.
getEncoding() on page 2-40	Returns the character encoding information, or the encoding information stored in the <?xml...?> tag or the user-defined output encoding if it has been more recently set.
getNodeValue() on page 2-40	Returns the value of this node.
getStandalone() on page 2-40	Returns the standalone information, or the standalone attribute stored in the <?xml...?> tag.
getVersion() on page 2-40	Retrieves the version information, or the version number stored in the <?xml...?> tag.
readExternal() on page 2-40	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:
setEncoding() on page 2-41	Sets the character encoding for output.
setStandalone() on page 2-41	Sets the standalone information stored in the <?xml...?> tag
setVersion() on page 2-41	Sets the version number stored in the <?xml...?> tag.
writeExternal() on page 2-42	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLDeclPI()

Creates a new instance of XMLDeclPI. The options are described in the following table.

Syntax	Description
`public XMLDeclPI();`	Creates a new instance of XMLDeclPI; default constructor.
`public XMLDeclPI( String version, String encoding, String standalone, boolean textDecl)`	Creates a new instance of XMLDeclP using version, encoding, standalone, and textDecl information.

Parameter	Description
`version`	The version used in creating a new XMLDeclPI.
`encoding`	The encoding used in creating a new XMLDeclPI.
`standaolone`	Specifies if the new XMLDeclPI is standalone.
`textDecl`	the text declaration of the new XMLDeclPI.

cloneNode()

Returns a duplicate of this node; serves as a generic copy constructor.

Syntax

```
public org.w3c.dom.Node cloneNode( boolean deep);
```

Parameter	Description
`deep`	If `true`, recursively clone the subtree under the specified node; if `false`, clone only the node itself (and its attributes, if it is an `Element`).

getData()

Returns the fully constructed string 'version=1.0....' Throws `DOMException`:

- DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

Syntax

```
public String getData();
```

getEncoding()

Returns the character encoding information, or the encoding information stored in the <?xml...?> tag or the user-defined output encoding if it has been more recently set.

Syntax

```
public final String getEncoding();
```

getNodeValue()

Returns the value of this node. Throws `DOMException`:

- `DOMSTRING_SIZE_ERR` raised when it would return more characters than fit in a `DOMString` variable on the implementation.

Syntax

```
public String getNodeValue();
```

getStandalone()

Returns the standalone information, or the standalone attribute stored in the <?xml...?> tag.

Syntax

```
public final String getStandalone();
```

getVersion()

Retrieves the version information, or the version number stored in the <?xml...?> tag.

Syntax

```
public final String getVersion();
```

readExternal()

Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:

- `IOException` when there is an error in reading the input stream.

- `ClassNotFoundException` when the class is not found

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
`in`	The ObjectInput stream used for reading the compressed stream.

setEncoding()

Sets the character encoding for output. Eventually, it sets the `ENCODING` stored in the <?xml...?> tag, but not until the document is saved. You should not call this method until the Document has been loaded.

Syntax

```
public final void setEncoding( String encoding);
```

Parameter	Description
`encoding`	The character encoding to set.

setStandalone()

Sets the standalone information stored in the <?xml...?> tag

Syntax

```
public final boolean setStandalone( String value);
```

Parameter	Description
`value`	Specifies if the XMLDeclPI class is standalone: `TRUE` for yes or `FALSE` for no.

setVersion()

Sets the version number stored in the <?xml...?> tag.

Syntax

```
public final void setVersion( String version);
```

Parameter	Description
version	The version information to set.

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws `IOException` when there is an exception while writing the compressed stream.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
out	The ObjectOutput stream used to write the compressed stream.

XMLDocument Class

This class implements the DOM Document interface, represents an entire XML document and serves the root of the Document Object Model tree. Each XML tag can either represent a node or a leaf of this tree.

According to the XML specification, the root of the tree consists of any combination of comments and processing instructions, but only one root element. A helper method `getDocumentElement` is provided as a short cut to finding the root element.

Syntax

```
public class XMLDocument implements java.io.Externalizable
```

Table 2–11 Summary of Methods of XMLDocument

Method	Description
XMLDocument() on page 2-46	Creates an empty document.
addID() on page 2-46	Adds an ID Element associated with this document.
adoptNode() on page 2-47	Adopts a node from another document to this document.
appendChild() on page 2-47	Appends a new node to the document. T
cloneNode() on page 2-47	Returns a duplicate of this node; serves as a generic copy constructor for nodes
createAttribute() on page 2-48	Creates an Attr of the given name.
createAttributeNS() on page 2-48	Creates an attribute with the given qualified name and namespace URI.
createCDATASection() on page 2-49	Creates a CDATASection node whose value is the specified string.
createComment() on page 2-49	Creates a Comment node given the specified string.
createDocumentFragment() on page 2-49	Creates an empty DocumentFragment object.
createElement() on page 2-50	Creates an element of the type specified.
createElementNS() on page 2-50	Creates an element of the given qualified name and namespace URI.

Table 2–11 (Cont.) Summary of Methods of XMLDocument

Method	Description
createEntityReference() on page 2-51	Creates an EntityReference object.
createEvent() on page 2-51	Creates an event object of the specified type.
createMutationEvent() on page 2-51	Creates a Mutation Event object of specified type.
createNodeIterator() on page 2-52	Creates a Node Iterator with specified root, flag which governs what type of nodes it should include in logical view, filter for filtering nodes, flag determining whether entity references and its descendants could be included.
createProcessingInstruction() on page 2-52	Creates a ProcessingInstruction node given the specified name and data strings. Throws DOMException:
createRange() on page 2-53	Creates a new Document Range Object, with Start and End Boundary points at the beginning of the document.
createRangeEvent() on page 2-53	Creates a Range Event object of specified type.
createTextNode() on page 2-53	Creates a Text node given the specified string.
createTraversalEvent() on page 2-53	Creates a Traversal Event object of specified type.
createTreeWalker() on page 2-54	Creates a Node Iterator with specified root, flag which governs what type of nodes it should include in logical view, filter for filtering nodes, flag determining whether entity references and its descendants could be included.
expectedElements() on page 2-54	Returns vector of element names that can be appended to the element.
getColumnNumber() on page 2-55	Returns column number debug information.
getDebugMode() on page 2-55	Returns the debug flag.
getDoctype() on page 2-55	Returns the Document Type Declaration (DTD) associated with this document.
getDocumentElement() on page 2-55	Accesses the child node that is the root element of the document.
getElementById() on page 2-55	Returns the Element whose ID is given by elementId.
getElementsByTagName() on page 2-56	Returns a NodeList of all the Elements with a given tag name in the order in which they would be encountered in a preorder traversal of the Document tree.

Table 2–11 (Cont.) Summary of Methods of XMLDocument

Method	Description
getElementsByTagNameNS() on page 2-56	Returns a NodeList of all the Elements with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of the Document tree.
getEncoding() on page 2-56	Returns the character encoding information stored in the <?xml...?> tag or the user-defined output encoding if it has been more recently set.
getIDHashtable() on page 2-57	Returns the ID element hashtable in the XML DOM Tree.
getImplementation() on page 2-57	Returns the DOMImplementation object that handles this document.
getLineNumber() on page 2-57	Returns line number debug information.
getNodeType() on page 2-57	Returns a code representing the type of the underlying object.
getOwnerDocument() on page 2-57	Returns the Document object associated with this node.
getStandalone() on page 2-57	Retrieves the standalone information; this is the standalone attribute stored in the <?xml...?> tag.
getSystemId() on page 2-58	Returns the system id of the entity contain this node.
getText() on page 2-58	Returns the non-marked-up text contained by this element.
getVersion() on page 2-58	Retrieves the version information stored in the <?xml...?> tag.
importNode() on page 2-58	Imports a node from another document to this document.
insertBefore() on page 2-59	Inserts the node newChild before the existing child node refChild.
print() on page 2-59	Writes the contents of this document to the given output.
printExternalDTD() on page 2-60	Writes the contents of this document to the given output stream.
readExternal() on page 2-61	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
removeChild() on page 2-61	Removes the elem from this documents list of child nodes.
replaceChild() on page 2-62	Replaces the child node oldChild with newChild in the list of children, and returns the oldChild node.

Table 2–11 (Cont.) Summary of Methods of XMLDocument

Method	Description
reportSAXEvents() on page 2-62	Reports SAX Events from a DOM Tree.
setDoctype() on page 2-62	Sets the doctype URI for the document.
setEncoding() on page 2-63	Sets the character encoding for output.
setLocale() on page 2-63	Sets the locale for error reporting.
setNodeContext() on page 2-63	Sets node context.
setParsedDoctype() on page 2-64	Sets the doctype object by parsing sysid.
setStandalone() on page 2-64	Sets the standalone information stored in the <?xml...?> tag.
setVersion() on page 2-64	Sets the version number stored in the <?xml...?> tag.
validateElementContent() on page 2-65	Validates the content of a element node.
writeExternal() on page 2-65	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLDocument()

Creates an empty document.

Syntax

```
public  XMLDocument();
```

addID()

Adds an ID Element associated with this document.

Syntax

```
public void addID( String name,
                   XMLElement e);
```

Parameter	Description
id	The id value.
e	XMLElement associated with id.

adoptNode()

Adopts a node from another document to this document. The returned node has no parent; parentNode is null. The source node is removed from the original document. Throws `DOMException`

- `NOT_SUPPORTED_ERR` raised if the type of the node being adopted is not supported.

Syntax

```
public org.w3c.dom.Node adoptNode( org.w3c.dom.Node srcNode);
```

Parameter	Description
`srcNode`	Node to be adopted.

appendChild()

Appends a new node to the document. Throws `DOMException`:

- `HIERARCHY_REQUEST_ERR` raised if this node is of a type that does not allow children of the type of the `elem` node.
- `WRONG_DOCUMENT_ERR` raised if `elem` was created from a different document than this one.

Syntax

```
public org.w3c.dom.Node appendChild( org.w3c.dom.Node newNode);
```

Parameter	Description
`newNode`	The new node to be added.

cloneNode()

Returns a duplicate of this node; serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns `null`.). Cloning an `Element` copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text

it contains unless it is a deep clone, since the text is contained in a child `Text` node. Cloning any other type of node simply returns a copy of this node.

Syntax

```
public org.w3c.dom.Node cloneNode( boolean deep);
```

Parameter	Description
`deep`	If `true`, recursively clone the subtree under the specified node; if `false`, clone only the node itself (and its attributes, if it is an `Element`).

createAttribute()

Creates an `Attr` of the given name. Note that the `Attr` instance can then be set on an `Element` using the `setAttribute` method. Throws `DOMException`:

- `INVALID_CHARACTER_ERR` raised if the specified name contains an invalid character.

Syntax

```
public org.w3c.dom.Attr createAttribute( String name);
```

Parameter	Description
`name`	The name of the new attribute.

createAttributeNS()

Creates an attribute with the given qualified name and namespace URI. Throws `DOMException`:

- `INVALID_CHARACTER_ERR` raised if the specified qualified name contains illegal Characters.
- `NAMESPACE_ERR` raised if the qualified name is malformed, if the qualified name has a prefix and the namespace URI is null or an empty string, or if the qulaifiedName has a prefix that is "xml" and namespace URI is different from "http://www.w3.org/2000/xmlns/"

Syntax

```
public org.w3c.dom.Attr createAttributeNS( String namespaceURI,
```

```
                                                    String qualifiedName);
```

Parameter	Description
namespaceURI	Namespace of the attribute/element to be created.
qualifiedName	Qualified name of the attribute/element to be created.

createCDATASection()

Creates a CDATASection node whose value is the specified string. Throws DOMException.

Syntax

```
public org.w3c.dom.CDATASection createCDATASection( String data);
```

Parameter	Description
data	The data for the CDATASection contents.

createComment()

Creates a Comment node given the specified string.

Syntax

```
public org.w3c.dom.Comment createComment( String data);
```

Parameter	Description
data	The data for the node.

createDocumentFragment()

Creates an empty DocumentFragment object.

Syntax

```
public org.w3c.dom.DocumentFragment createDocumentFragment();
```

createElement()

Creates an element of the type specified. Note that the instance returned implements the Element interface, so attributes can be specified directly on the returned object. Throws `DOMException`:

- `INVALID_CHARACTER_ERR` raised if the specified name contains an invalid character.

Syntax

```
public org.w3c.dom.Element createElement( String tagName);
```

Parameter	Description
`tagName`	The name of the element type to instantiate. The name is treated as case-sensitive.

createElementNS()

Creates an element of the given qualified name and namespace URI. Throws `DOMException`:

- `INVALID_CHARACTER_ERR` raised if the specified qualified name contains illegal Characters.
- `NAMESPACE_ERR` raised if the qualified name is malformed, if the qualified name has a prefix and the namespace URI is null or an empty string, or if the qulaifiedName has a prefix that is "xml" and namespace URI is different from "http://www.w3.org/XML/1998/namespace".

Syntax

```
public org.w3c.dom.Element createElementNS( String namespaceURI,
                                             String qualifiedName);
```

Parameter	Description
`namespaceURI`	Namespace of the attribute/element to be created.
`qualifiedName`	Qualified name of the attribute/element to be created.

createEntityReference()

Creates an EntityReference object. Throws `DOMException`:

- `INVALID_CHARACTER_ERR` raised if the specified name contains an invalid character.

Syntax

```
public org.w3c.dom.EntityReference createEntityReference( String name);
```

Parameter	Description
`name`	The name of the entity to reference.

createEvent()

Creates an event object of the specified type.

Syntax

```
public org.w3c.dom.events.Event createEvent( String type);
```

Parameter	Description
`type`	The type of the event.

createMutationEvent()

Creates a Mutation Event object of specified type.

Syntax

```
public org.w3c.dom.events.MutationEvent createMutationEvent( String type);
```

Parameter	Description
`type`	The type of the mutation event.

createNodeIterator()

Creates a Node Iterator with specified root, flag which governs what type of nodes it should include in logical view, filter for filtering nodes, flag determining whether entity references and its descendants could be included. Throws `DOMException`:

- `NOT_SUPPORTED_ERR` if the NodeIterator could not be created with specified root.

Syntax

```
public org.w3c.dom.traversal.NodeIterator createNodeIterator(
                           org.w3c.dom.Node root,
                           int whatToShow,
                           org.w3c.dom.traversal.NodeFilter filter,
                           boolean expandEntityReferences);
```

Parameter	Description
`root`	Root node of the iterator.
`whatToShow`	Flag indicating what type of nodes will be included in the iterator/tree walker.
`filter`	NodeFilter to filter unwanted nodes from the iterator/tree walker.
`expandEntityReference`	Flag to indicate traversal of entity references.

createProcessingInstruction()

Creates a `ProcessingInstruction` node given the specified name and data strings. Throws `DOMException`:

- `INVALID_CHARACTER_ERR`: Raised if an invalid character is specified.

Syntax

```
public org.w3c.dom.ProcessingInstruction createProcessingInstruction(
                     String target,
                     String data);
```

Parameter	Description
`target`	The target part of the processing instruction.

Parameter	Description
data	The data for the node.

createRange()

Creates a new Document Range Object, with Start and End Boundary points at the beginning of the document.

Syntax

```
public org.w3c.dom.ranges.Range createRange();
```

createRangeEvent()

Creates a Range Event object of specified type.

Syntax

```
public org.w3c.dom.events.Event createRangeEvent( String type);
```

Parameter	Description
type	The type of the Range event.

createTextNode()

Creates a `Text` node given the specified string.

Syntax

```
public org.w3c.dom.Text createTextNode( String data);
```

Parameter	Description
data	The data of the node.

createTraversalEvent()

Creates a Traversal Event object of specified type.

Syntax

```
public org.w3c.dom.events.Event createTraversalEvent( String type);
```

Parameter	Description
type	The type of the traversal event.

createTreeWalker()

Creates a Node Iterator with specified root, flag which governs what type of nodes it should include in logical view, filter for filtering nodes, flag determining whether entity references and its descendants could be included. Throws DOMException:

- NOT_SUPPORTED_ERR if NodeIterator cannot be created with specified root.

Syntax

```
public org.w3c.dom.traversal.TreeWalker createTreeWalker(
                          org.w3c.dom.Node root,
                          int whatToShow,
                          org.w3c.dom.traversal.NodeFilter filter,
                          boolean expandEntityReferences);
```

Parameter	Description
root	Root node of the iterator.
whatToShow	Flag indicating what type of nodes will be included in the iterator/tree walker.
filter	NodeFilter to filter unwanted nodes from the iterator/tree walker.
expandEntityReference	Flag to indicate traversal of entity references.

expectedElements()

Returns vector of element names that can be appended to the element.

Syntax

```
public java.util.Vector expectedElements( org.w3c.dom.Element e);
```

Parameter	Description
e	Element

getColumnNumber()

Returns column number debug information.

Syntax

```
public int getColumnNumber();
```

getDebugMode()

Returns the debug flag.

Syntax

```
public boolean getDebugMode();
```

getDoctype()

Returns the Document Type Declaration (DTD) associated with this document. For XML documents without a DTD, this returns `null`. Note that the DOM Level 1 specification does not support editing the DTD.

Syntax

```
public org.w3c.dom.DocumentType getDoctype();
```

getDocumentElement()

Accesses the child node that is the root element of the document.

Syntax

```
public org.w3c.dom.Element getDocumentElement();
```

getElementById()

Returns the Element whose ID is given by elementId. If no such element exists, returns null. Behavior is not defined if more than one element has this ID.

Syntax

```
public org.w3c.dom.Element getElementById( String elementId);
```

Parameter	Description
elementId	The elementId used to get the matching Id Element.

getElementsByTagName()

Returns a NodeList of all the Elements with a given tag name in the order in which they would be encountered in a preorder traversal of the Document tree.

Syntax

```
public org.w3c.dom.NodeList getElementsByTagName( String tagname);
```

Parameter	Description
tagname	The name of the tag to match on. The special value "*" matches all tags.

getElementsByTagNameNS()

Returns a NodeList of all the Elements with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of the Document tree.

Syntax

```
public org.w3c.dom.NodeList getElementsByTagNameNS( String namespaceURI,
                                                    String localName);
```

Parameter	Description
namespaceURI	Namespace of the elements requested.
localName	Local name of the element requested.

getEncoding()

Returns the character encoding information stored in the <?xml...?> tag or the user-defined output encoding if it has been more recently set.

Syntax

```
public final String getEncoding();
```

getIDHashtable()

Returns the ID element hashtable in the XML DOM Tree.

Syntax

```
public java.util.Hashtable getIDHashtable();
```

getImplementation()

Returns the `DOMImplementation` object that handles this document. A DOM application may use objects from multiple implementations.

Syntax

```
public org.w3c.dom.DOMImplementation getImplementation();
```

getLineNumber()

Returns line number debug information.

Syntax

```
public int getLineNumber();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

getOwnerDocument()

Returns the `Document` object associated with this node. Since this node is a `Document`, this is `null`.

Syntax

```
public org.w3c.dom.Document getOwnerDocument();
```

getStandalone()

Retrieves the standalone information; this is the standalone attribute stored in the <?xml...?> tag.

Syntax

```
public final String getStandalone();
```

getSystemId()

Returns the system id of the entity contain this node.

Syntax

```
public String getSystemId();
```

getText()

Returns the non-marked-up text contained by this element. For text elements, this is the raw data. For elements with child nodes, this method traverses the entire subtree and appends the text for each terminal text element, effectively stripping out the XML markup for the subtree. For example, if the XML document contains "William Shakespeare", `XMLDocument.getText` returns "William Shakespeare".

Syntax

```
public String getText();
```

getVersion()

Retrieves the version information stored in the <?xml...?> tag.

Syntax

```
public final String getVersion();
```

importNode()

Imports a node from another document to this document. The returned node has no parent; (parentNode is null). The source node is not altered or removed from the original document. For all nodes, importing a node creates a node object owned by the importing document, with attribute values identical to the source node's nodeName and nodeType, plus the attributes related to namespaces (prefix, localName, and namespaceURI). Throws `DOMException`:

- `NOT_SUPPORTED_ERR` raised if the type of the node being imported is not supported.

Syntax

```
public org.w3c.dom.Node importNode( org.w3c.dom.Node importedNode,
                                    boolean deep);
```

Parameter	Description
`importedNode`	Node to be imported.

Parameter	Description
deep	Indicates whether the descen.dants of this node are to be imported.

insertBefore()

Inserts the node newChild before the existing child node refChild. If refChild is null, insert newChild at the end of the list of children. If newChild is a DocumentFragment object, all of its children are inserted, in the same order, before refChild. If the newChild is already in the tree, it is first removed. Throws DOMException:

- HIERARCHY_REQUEST_ERR raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to insert is one of this node's ancestors.
- WRONG_DOCUMENT_ERR raised if newChild was created from a different document than the one that created this node.
- NO_MODIFICATION_ALLOWED_ERR raised if this node is readonly.
- NOT_FOUND_ERR raised if refChild is not a child of this node.

Syntax

```
public org.w3c.dom.Node insertBefore( org.w3c.dom.Node newChild,
                                      org.w3c.dom.Node refChild);
```

Parameter	Description
newChild	The node to insert.
refChild	The reference node, or the node before which the new node must be inserted.

print()

Writes the contents of this document to the given output. Throws IOException. The options are described in the following table.

Syntax	Description
`public void print( java.io.OutputStream out);`	Writes the contents of this document to the given output stream.
`public void print( java.io.OutputStream out, String enc);`	Writes the contents of this document to the given encoded output stream.
`public void print( java.io.PrintWriter out);`	Writes the contents of this document to the given print writer.
`public void print( PrintDriver pd);`	Writes the contents of this document to the given print driver.

Parameter	Description
`out`	Output to write to.
`enc`	Encoding to use for the output.
`pd`	PrintDriver used to write each node.

printExternalDTD()

Writes the contents of this document to the given output stream. Throws `IOException`. The options are described in the following table.

Syntax	Description
`public void printExternalDTD( java.io.OutputStream out);`	Writes the contents of this document to the given output stream.
`public void printExternalDTD( java.io.OutputStream out, String enc);`	Writes the contents of this document to the given encoded output stream.
`public void printExternalDTD( java.io.PrintWriter out);`	Writes the contents of this document to the given print writer.

Parameter	Description
out	the output to write to.
enc	Encoding to use for the output.

readExternal()

Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:

- IOException thrown when there is an error in reading the input stream.
- ClassNotFoundException thrown when the class is not found.

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
in	The ObjectInput stream used for reading the compressed stream.

removeChild()

Removes the elem from this documents list of child nodes. Throws DOMException:

- NO_MODIFICATION_ALLOWED_ERR raised if this document is readonly.
- NOT_FOUND_ERR raised if oldChild is not a child of this node.

Syntax

```
public org.w3c.dom.Node removeChild( org.w3c.dom.Node elem);
```

Parameter	Description
elem	The element to be removed.

replaceChild()

Replaces the child node `oldChild` with `newChild` in the list of children, and returns the `oldChild` node. If the `newChild` is already in the tree, it is first removed. Throws `DOMException`:

- `HIERARCHY_REQUEST_ERR` raised if this node is of a type that does not allow children of the type of the `newChild` node.
- `WRONG_DOCUMENT_ERR` raised if `newChild` was created from a different document than this one.
- `NOT_FOUND_ERR` raised if `oldChild` is not a child of this node.

Syntax

```
public org.w3c.dom.Node replaceChild( org.w3c.dom.Node newChild,
                                      org.w3c.dom.Node oldChild);
```

Parameter	Description
`newChild`	The new node to put in the child list.
`oldChild`	he node being replaced in the list.

reportSAXEvents()

Reports SAX Events from a DOM Tree. Throws a `SAXException`.

Syntax

```
public void reportSAXEvents( org.xml.sax.ContentHandler cntHandler);
```

Parameter	Description
`cntHandler`	The content handler.

setDoctype()

Sets the doctype URI for the document.

Syntax

```
public void setDoctype( String rootname,
```

```
                                String sysid,
                                String pubid);
```

Parameter	Description
root	The name of the root element.
sysid	The system id of the doctype.
pubid	The public id of the doctype (can be null).

setEncoding()

Sets the character encoding for output. Eventually it sets the ENCODING stored in the <?xml...?> tag, but not until the document is saved. This method should not be called until the Document has been loaded.

Syntax

```
public final void setEncoding( String encoding);
```

Parameter	Description
encoding	The character encoding to set.

setLocale()

Sets the locale for error reporting.

Syntax

```
public final void setLocale( java.util.Locale locale);
```

Parameter	Description
locale	Locale for error reporting.

setNodeContext()

Sets node context.

Syntax

```
public void setNodeContext( oracle.xml.util.NodeContext nctx);
```

Parameter	Description
nctx	The context to set.

setParsedDoctype()

Sets the doctype object by parsing sysid.

Syntax

```
public void setParsedDoctype( String rootname,
                              String sysid,
                              String pubid);
```

Parameter	Description
root	The name of the root element.
sysid	The system id of the doctype.
pubid	The public id of the doctype (can be null),.

setStandalone()

Sets the standalone information stored in the <?xml...?> tag.

Syntax

```
public final void setStandalone( String value);
```

Parameter	Description
value	The attribute value.

setVersion()

Sets the version number stored in the <?xml...?> tag.

Syntax

```
public final void setVersion( String version);
```

Parameter	Description
version	The version information to set.

validateElementContent()

Validates the content of a element node. Returns TRUE if valid, FALSE otherwise.

Syntax

```
public boolean validateElementContent( org.w3c.dom.Element elem);
```

Parameter	Description
elem	Element to be validated.

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws IOException when there is an exception while writing the serialized/compressed stream.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
out	The ObjectOutput stream used to write the serialized/ compressed stream.

XMLDocumentFragment Class

This class implements the DOM DocumentFragment interface. Extends XMLElement rather than XMLNode so it can be handled as an element. See also `DocumentFragment, NodeFactory, DOMParser.setNodeFactory()`.

Syntax

```
public class XMLDocumentFragment implements java.io.Serializable
```

Table 2–12 Summary of Methods of XMLDocumentFragment

Method	Description
getAttributes() on page 2-66	Returns the attributes of the XMLDocumentFragment.
getNodeType() on page 2-66	Returns a code representing the type of the underlying object.
getParentNode() on page 2-66	Returns the parent of this node.

getAttributes()

Returns the attributes of the XMLDocumentFragment - an empty NamedNodeMap.

Syntax

```
public org.w3c.dom.NamedNodeMap getAttributes();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

getParentNode()

Returns the parent of this node.

Syntax

```
public org.w3c.dom.Node getParentNode();
```

XMLDOMException Class

This class is used to throw DOM exceptions.

Syntax

```
public class XMLDOMException
```

XMLDOMException()

Constructs an `XMLDOMException` exception. The options are described in the following table.

Syntax	Description
`public XMLDOMException( short code);`	Constructs an `XMLDOMException` exception with a specified code.
`public XMLDOMException( short code, String mess);`	Constructs an `XMLDOMException` exception with a specified message and an error code.

Parameter	Description
`code`	Code indicated in DOM interface, uses default message
`mess`	Message used in constructing the XMLDOMException

XMLDOMImplementation

This class implements the DOMImplementation

Syntax

```
public class XMLDOMImplementation implements java.io.Serializable
```

Table 2–13 Summary of Methods of XMLDOMImplementation

Method	Description
XMLDOMImplementation() on page 2-68	Creates a new instance of XMLDOMImplementation.
createDocument() on page 2-68	Creates an XMLDocument object containing the specified DocumentType Node and a root element with the specified names and the empty DocumentType node.
createDocumentType() on page 2-69	Creates an empty DocumentType node with root element name and system/public identifier.
hasFeature() on page 2-70	Tests if the DOM implementation implements a specific feature.
setFeature() on page 2-70	Sets a specified feature.

XMLDOMImplementation()

Creates a new instance of XMLDOMImplementation.

Syntax

```
public  XMLDOMImplementation();
```

createDocument()

Creates an XMLDocument object containing the specified DocumentType Node and a root element with the specified names and the empty DocumentType node. Throws `DOMException`:

- `INVALID_CHARACTER_ERR` raised if the specified qualified name contains an illegal character.
- `NAMESPACE_ERR` raised if the qualifiedName is malformed, if the qualifiedName has a prefix and the namespaceURI is null or an empty String,

or if the qualifiedName has a prefix that is "xml" and namespaceURI is different from "http://www.w3.org/XML/1998/namespace"

- WRONG_DOCUMENT_ERR raised if doctype has already been used with a different document or was created from a different implementation.

Syntax

```
public org.w3c.dom.Document createDocument( String namespaceURI,
                                            String qualifiedName,
                                            org.w3c.dom.DocumentType doctype);
```

Parameter	Description
namespaceURI	Namespace of the root element in the document.
qualifiedName	Qualified name of the root element in the document.
doctype	DocumentType (DTD) associated with the document.

createDocumentType()

Creates an empty DocumentType node with root element name and system/public identifier. Returns the DocumentType object created. Throws DOMException:

- INVALID_CHARACTER_ERR raised if the specified qualified name contains an illegal character.
- NAMESPACE_ERR raised if the qualifiedName is malformed.

Syntax

```
public org.w3c.dom.DocumentType createDocumentType( String qualifiedName,
                                                    String publicId,
                                                    String systemId);
```

Parameter	Description
qualifiedName	Qualified name of the root element.
systemid	System identifier of the DocumentType node.
publicid	Public identifier of the DocumentType node.

hasFeature()

Tests if the DOM implementation implements a specific feature. Returns TRUE if the feature is implemented, FALSE otherwise.

Syntax

```
public boolean hasFeature( String feature,
                           String version);
```

Parameter	Description
`feature`	The feature being tested.
`version`	The version of the feature being tested.

setFeature()

Sets a specified feature. Throws a `DOMException` if the feature could not be set.

Syntax

```
public void setFeature( String feature);
```

Parameter	Description
`feature`	The DOM feature.

XMLElement Class

This class implements the DOM `Element` Interface.

Syntax

```
public class XMLElement implements oracle.xml.parser.v2.NSName,
oracle.xml.parser.v2.NSResolver, java.io.Externalizable
```

Table 2–14 Summary of Methods of XMLElement

Method	Description
XMLElement() on page 2-73	Default constructor.
cloneNode() on page 2-73	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
getAttribute() on page 2-74	Returns an attribute value by name; if that attribute does not have a specified or default value, returns an empty string.
getAttributeNode() on page 2-74	Returns an Attr node by name, or NULL if there is no such attribute.
getAttributeNodeNS() on page 2-74	Returns attribute with the given namespaceURI and localName if it exists; otherwise, returns NULL.
getAttributeNS() on page 2-75	Returns the value of the attribute with namespace URI and localName, if it exists; threshed, returns NULL.
getAttributes() on page 2-75	Returns a NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.
getChildrenByTagName() on page 2-75	Returns a NodeList of all immediate children with a given tag name. The options are described in the following table.
getElementsByTagName() on page 2-76	Returns a NodeList of all the Elements with a given tag name in the order in which they would be encountered in a preorder traversal of the Document tree.
getElementsByTagNameNS( ) on page 2-76	Returns a NodeList of all the descendant Elements with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of this Element tree.
getExpandedName() on page 2-76	Returns the fully resolved name for this element.

Table 2–14 (Cont.) Summary of Methods of XMLElement

Method	Description
getFirstAttribute() on page 2-74	Retrieves the first Attr node, or NULL if there is no attribute.
getLocalName() on page 2-77	Returns the local Name for this element.
getNamespaceURI() on page 2-77	Returns the name space URI of this element.
getNodeType() on page 2-77	Returns a code representing the type of the underlying object.
getPrefix() on page 2-77	Returns the namespace prefix for this element.
getQualifiedName() on page 2-77	Returns the qualified name for this element.
getTagName() on page 2-78	Returns the name of the element.
hasAttribute() on page 2-78	Returns TRUE when an attribute with a given name is specified on this element or has a default value.
hasAttributeNS() on page 2-78	Returns TRUE when an attribute with a given local name and namespace URI is specified on this element or has a default value.
hasAttributes() on page 2-78	Returns TRUE if this node has any attributes.
readExternal() on page 2-79	Restores the information written by `writeExternal()` by reading the input stream and regenerating the objects according to the information of the input stream.
removeAttribute() on page 2-79	Removes an attribute by name.
removeAttributeNode() on page 2-79	Removes and returns the specified attribute.
removeAttributeNS() on page 2-80	Removes an attribute by local name and namespace URI.
reportSAXEvents() on page 2-80	Reports SAX Events from a DOM Tree.
resolveNamespacePrefix() on page 2-80	Finds the namespace definition in scope in this element, given a namespace prefix.
setAttribute() on page 2-81	Adds a new attribute.
setAttributeNode() on page 2-81	Adds a new attribute node.

Table 2–14 (Cont.) Summary of Methods of XMLElement

Method	Description
setAttributeNodeNS() on page 2-82	Adds a new namespace aware attribute node.
validateContent() on page 2-83	Validates the content of a element node.
writeExternal() on page 2-84	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLElement()

Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node. In order to deserialize this node to construct the DOM node from the serialized/ compressed stream, it is required to create a handle of the object. For all normal XMLElement creation use `createElement()` of XMLDocument.

Syntax

```
public  XMLElement();
```

cloneNode()

Returns a duplicate of this node; serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns `null`.). Cloning an `Element` copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child `Text` node. Cloning any other type of node simply returns a copy of this node.

Syntax

```
public org.w3c.dom.Node cloneNode(boolean deep);
```

Parameter	Description
`true`	If `true`, recursively clone the subtree under the specified node; if `false`, clone only the node itself (and its attributes, if it is an `Element`).

getAttribute()

Returns an attribute value by name; if that attribute does not have a specified or default value, returns an empty string.

Syntax

```
public String getAttribute( String name);
```

Parameter	Description
name	The name of the attribute to retrieve.

getAttributeNode()

Returns an Attr node by name, or NULL if there is no such attribute.

Syntax

```
public org.w3c.dom.Attr getAttributeNode( String name);
```

Parameter	Description
name	The name of the attribute node to retrieve.

getAttributeNodeNS()

Returns attribute with the given namespaceURI and localName if it exists; otherwise, returns NULL.

Syntax

```
public org.w3c.dom.Attr getAttributeNodeNS( String namespaceURI,
                                            String localName);
```

Parameter	Description
namespaceURI	Namespace of the attribute node requested.
localName	Local name of the attribute node requested.

getAttributeNS()

Returns the value of the attribute with namespace URI and localName, if it exists; threshed, returns NULL.

Syntax

```
public String getAttributeNS( String namespaceURI,
                              String localName);
```

Parameter	Description
namespaceURI	Namespace of the attribute requested.
localName	Local name of the attribute requested.

getAttributes()

Returns a NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

Syntax

```
public org.w3c.dom.NamedNodeMap getAttributes();
```

getChildrenByTagName()

Returns a NodeList of all immediate children with a given tag name. The options are described in the following table.

Syntax	Description
public org.w3c.dom.NodeList getChildrenByTagName(String name);	Returns a NodeList of all immediate children with a given tag name.
public org.w3c.dom.NodeList getChildrenByTagName(String name, String ns);	Returns a NodeList of all immediate children with a given tag name and namespace.

Parameter	Description
name	The name of the tag to match on (should be local name).
ns	The name space.

getElementsByTagName()

Returns a NodeList of all the Elements with a given tag name in the order in which they would be encountered in a preorder traversal of the Document tree.

Syntax

```
public org.w3c.dom.NodeList getElementsByTagName( String tagname);
```

Parameter	Description
tagname	The name of the tag to match on. The special value "*" matches all tags.

getElementsByTagNameNS()

Returns a NodeList of all the descendant Elements with a given local name and namespace URI in the order in which they are encountered in a preorder traversal of this Element tree.

Syntax

```
public org.w3c.dom.NodeList getElementsByTagNameNS( String namespaceURI,
                                                    String localName);
```

Parameter	Description
namespaceURI	The namespace of the element.
localName	The local name of the element.

getExpandedName()

Returns the fully resolved name for this element.

Syntax

```
public String getExpandedName();
```

getFirstAttribute()

Retrieves the first `Attr` node, or `NULL` if there is no attribute.

Syntax

```
public XMLNode getFirstAttribute();
```

getLocalName()

Returns the local Name for this element.

Syntax

```
public String getLocalName();
```

getNamespaceURI()

Returns the name space URI of this element.

Syntax

```
public String getNamespaceURI();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

getPrefix()

Returns the namespace prefix for this element.

Syntax

```
public String getPrefix();
```

getQualifiedName()

Returns the qualified name for this element.

Syntax

```
public String getQualifiedName();
```

getTagName()

Returns the name of the element. For example, in: <elementExample id="demo">... </elementExample>, `tagName` has the value `elementExample`. Note that this is case-preserving in XML, as are all of the operations of the DOM. The HTML DOM returns the `tagName` of an HTML element in the canonical uppercase form, regardless of the case in the source HTML document.

Syntax

```
public String getTagName();
```

hasAttribute()

Returns `TRUE` when an attribute with a given name is specified on this element or has a default value, `FALSE` otherwise.

Syntax

```
public boolean hasAttribute( String name);
```

Parameter	Description
`name`	Name of the attribute whose presence is checked

hasAttributeNS()

Returns `TRUE` when an attribute with a given local name and namespace URI is specified on this element or has a default value; returns `FALSE` otherwise.

Syntax

```
public boolean hasAttributeNS( String namespaceURI,
                               String localName);
```

Parameter	Description
`namespaceURI`	Namespace of the attribute whose presence is checked.
`localName`	Local name of the attribute whose presence is checked.

hasAttributes()

Returns `TRUE` if this node has any attributes, `FALSE` otherwise.

Syntax

```
public boolean hasAttributes();
```

readExternal()

Restores the information written by writeExternal by reading the input stream and regenerating the objects according to the information of the input stream. Throws the following exceptions:

- `IOException` when there is an exception reading the compressed stream.
- `ClassNotFoundException` when the class is not found

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
`in`	ObjectInput stream used to read the compressed stream.

removeAttribute()

Removes an attribute by name. If the removed attribute has a default value it is immediately replaced. Throws `DOMException`:

- `NO_MODIFICATION_ALLOWED_ERR` raised if this node is readonly.

Syntax

```
public void removeAttribute( String name);
```

Parameter	Description
`name`	Name of the attribute to remove.

removeAttributeNode()

Removes and returns the specified attribute. Throws `DOMException`:

- `NO_MODIFICATION_ALLOWED_ERR` raised if this node is readonly.
- `NOT_FOUND_ERR` raised if `oldAttr` is not an attribute of the element.

Syntax

```
public org.w3c.dom.Attr removeAttributeNode( org.w3c.dom.Attr oldAttr);
```

Parameter	Description
oldAttr	The Attr node to remove from the attribute list. If the removed Attr has a default value it is immediately replaced.

removeAttributeNS()

Removes an attribute by local name and namespace URI. Throws DOMEXception:

- NO_MODIFICATIONS_ALLOWED_ERR if this element is readonly.

Syntax

```
public void removeAttributeNS( String namespaceURI,
                               String localName);
```

Parameter	Description
namespaceURI	Namespace of the attribute to be removed.
localName	Local name of the attribute to be removed.

reportSAXEvents()

Report SAX Events from a DOM Tree. Throws SAXException.

Syntax

```
public void reportSAXEvents( org.xml.sax.ContentHandler cntHandler);
```

Parameter	Description
cntHandler	The Content Handler.

resolveNamespacePrefix()

Finds the namespace definition in scope in this element, given a namespace prefix.

Syntax

```
public String resolveNamespacePrefix( String prefix);
```

Parameter	Description
prefix	Namespace prefix to be resolved if the prefix; if default, returns the default namespace

setAttribute()

Adds a new attribute. If an attribute with that name is already present in the element, its value is changed to be that of the value parameter. This value is a simple string, it is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an `Attr` node plus any `Text` and `EntityReference` nodes, build the appropriate subtree, and use `setAttributeNode` to assign it as the value of an attribute. This method is namespace unaware and hence wont result in update of namespace table if a new attr is added through this method. Throws `DOMException`:

- `INVALID_CHARACTER_ERR` if the specified name contains an invalid character.
- `NO_MODIFICATION_ALLOWED_ERR` raised if this node is readonly.

Syntax

```
public void setAttribute( String name,
                          String value);
```

Parameter	Description
name	The name of the attribute to create or alter.
value	Value to set in string form.

setAttributeNode()

Adds a new attribute. If an attribute with that name is already present in the element, it is replaced by the new one. If the `newAttr` attribute replaces an existing

attribute with the same name, the previously existing `Attr` node is returned, otherwise `null` is returned. Throws `DOMException`:

- `WRONG_DOCUMENT_ERR` raised if `newAttr` was created from a different document than the one that created the element.
- `NO_MODIFICATION_ALLOWED_ERR` raised if this node is readonly.
- `INUSE_ATTRIBUTE_ERR` raised if `newAttr` is already an attribute of another `Element` object. The DOM user must explicitly clone `Attr` nodes to reuse them in other elements.

Syntax

```
public org.w3c.dom.Attr setAttributeNode ( org.w3c.dom.Attr newAttr);
```

Parameter	Description
`newAttr`	Attribute to be added to the attribute list.

setAttributeNodeNS()

Adds a new attribute. Throws `DOMException`:

- `INVALID_CHARACTER_ERR` if specified name contains illegal characters.
- `NAMESPACE_ERR` raised if the qualified name is malformed, if the qualified name has a prefix and the namespace URI is null or an empty string, or if the qulaifiedName is "xmlns" and namespace URI is different from "http://www.w3.org/2000/xmlns/", or if qualifiedName has a prefix that is "xml" and the namespaceURI is different from http://www.w3.org/XML/1998/namespaces.
- `NO_MODIFICATION_ALLOWED_ERR` raised if this node is readonly.
- `WRONG_DOCUMENT_ERR` raised if the newAttr was created from a document different from the one that created the document.
- `INUSE_ATTRIBUTE_ERR` raised if newAttr is already an attribute of another Element object.

The options are described in the following table.

Syntax	Description
`public org.w3c.dom.Attr setAttributeNodeNS( org.w3c.dom.Attr newAttr);`	Adds and returns a new attribute node. If an attribute with that local name and that namespace URI is already present in the element, it is replaced by the new one.
`public void setAttributeNS( String namespaceURI, String qualifiedName, String value);`	Constructs a new attribute node from `nemespaceURI`, `qualifiedName`, and `value`, and adds it. If an attribute with the same local name and namespace URI is already present on the element, its prefix is changed to be the prefix part of the qualifiedName, and its value is changed to be the value parameter. This value is a simple string; it is not parsed as it is being set. Therefore, any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out.

Parameter	Description
`newAttr`	Attribute to be added to the attribute list.
`namespaceURI`	Namespace of the attribute to be added.
`localName`	Local name of the attribute to be added.
`value`	Value of the attribute to be added.

validateContent()

Validates the content of a element node. Returns TRUE if valid, FALSE otherwise. The options are described in the following table.

Syntax	Description
`public boolean validateContent( DTD dtd);`	Validates the content of a element node using the DTD.
`public boolean validateContent( oracle.xml.parser.schema.XMLSchema schema);`	Validates the content of the element node against given XML Schema param schema.

Syntax	Description
`public boolean validateContent( oracle.xml.parser.schema.XMLSchema schema, String mode);`	Validates the content of the element against given XML Schema in the given mode.

Parameter	Description
`dtd`	The DTD object used to validate the element.
`schema`	The XMLSchema object used to validate the element.
`mode`	The validation mode.

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
`out`	The ObjectOutput stream used to write the serialized/compressed

XMLEntity Class

This class implements the DOM `Entity` interface and represents an XML internal or external entity as defined in the XML Document Type Definition (DTD).

Syntax

```
public class XMLEntity implements java.io.Externalizable
```

Table 2–15 Summary of Methods of XMLEntity

Method	Description
XMLEntity() on page 2-85	Default constructor.
cloneNode() on page 2-86	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
getNodeType() on page 2-86	Returns a code representing the type of the underlying object.
getNodeValue() on page 2-86	Returns the value of this node, depending on its type.
getNotationName() on page 2-86	Returns the name of the notation for an unparsed the entity. For parsed entities, this is `null`.
getPublicId() on page 2-87	Returns the public identifier.
getSystemId() on page 2-87	Returns the system identifier.
readExternal() on page 2-87	Reads the information written in the compressed stream by writeExternal() method and restores the object correspondingly.
setNodeValue() on page 2-87	Sets the value of entity.
writeExternal() on page 2-88	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLEntity()

Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node. In order to deserialize this node to construct the DOM node from the serialized/ compressed stream, it is required to create a handle of the object.

Syntax

```
public  XMLEntity();
```

cloneNode()

Returns a duplicate of this node; serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns `null`.). Cloning an `Element` copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child `Text` node. Cloning any other type of node simply returns a copy of this node.

Syntax

```
public org.w3c.dom.Node cloneNode(boolean deep);
```

Parameter	Description
`deep`	If `TRUE`, recursively clones the subtree under the specified node; if `FALSE`, clones only the node itself (and its attributes, if it is an `Element`).

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

getNodeValue()

Returns the value of this node, depending on its type. Throws `DOMException`:

- `NO_MODIFICATION_ALLOWED_ERR` raised when the node is readonly.
- `DOMSTRING_SIZE_ERR` raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

Syntax

```
public String getNodeValue();
```

getNotationName()

Returns the name of the notation for an unparsed the entity. For parsed entities, this is `null`.

Syntax

```
public String getNotationName();
```

getPublicId()

Returns the public identifier associated with the entity, if specified. If the public identifier was not specified, this is `null`.

Syntax

```
public String getPublicId();
```

getSystemId()

Returns the system identifier associated with the entity, if specified. If the system identifier was not specified, this is `null`.

Syntax

```
public String getSystemId();
```

readExternal()

Reads the information written in the compressed stream by writeExternal() method and restores the object correspondingly. Throws the following exceptions:

- `IOException` when there is an error in reading the input stream.
- `ClassNotFoundException` when the class is not found.

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
`in`	The ObjectInput stream used for reading the compressed stream.

setNodeValue()

Sets the value of entity.

Syntax

```
public void setNodeValue( String arg);
```

Parameter	Description
arg	The new value of the entity.

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws `IOException`.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
out	The ObjectOutput stream used to write the serialized/ compressed stream.

XMLEntityReference Class

This class implements DOM `EntityReference` interface.

Syntax

```
public class XMLEntityReference implements java.lang.Cloneable,
java.io.Externalizable
```

Table 2–16 Summary of Methods of XMLEntityReference

Method	Description
XMLEntityReference() on page 2-89	Default constructor.
getNodeType() on page 2-89	Returns a code representing the type of the underlying object.
readExternal() on page 2-90	Reads the information written in the compressed stream by writeExternal() method and restores the object correspondingly.
writeExternal() on page 2-90	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLEntityReference()

Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node. In order to deserialize this node to construct the DOM node from the serialized/ compressed stream, it is required to create a handle of the object.

Syntax

```
public  XMLEntityReference();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

readExternal()

Reads the information written in the compressed stream by writeExternal() method and restores the object correspondingly. Throws the following exceptions:

- `IOException` when there is an error in reading the input stream.
- `ClassNotFoundException` when the class is not found.

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
`in`	The ObjectInput stream used for reading the compressed stream

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws `IOException` when there is an exception while writing the compressed stream.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
`out`	The ObjectOutput stream used to write the compressed stream.

XMLNode Class

Implements the DOM `Node` interface and serves as the primary datatype for the entire Document Object Model. It represents a single node in the document tree.

The attributes nodeName, nodeValue and attributes are included as a mechanism to get at node information without casting down to the specific derived instance. In cases where there is no obvious mapping of these attributes for a specific nodeType (for example, nodeValue for an Element or attributes for a Comment), this returns null. Note that the derived classes may contain additional and more convenient mechanisms to get and set the relevant information. This DOM Nodes extending XMLNode instead of XMLNSNode have fixed Nodename defined by DOM specification. Also only node that cannot have child nodes extend this class.

Syntax

```
public abstract class XMLNode implements java.lang.Cloneable,
java.io.Externalizable
```

Table 2–17 Fields of XMLNode

Field	Syntax	Description
`ATTRDECL`	`public static final short ATTRDECL`	An attribute declaration node.
`Auto_Events`	`public static final String Auto_Events`	Flag to set Auto EVENTS.
`capturing`	`public static final String capturing`	Can be handled by one of the event's target's ancestors before being handled by the event's target.
`DOMAttrModified`	`public static final String DOMAttrModified`	Attr has been modified on a node.
`DOMCharacterDataModified`	`public static final String DOMCharacterDataModified`	Characterized within a node has been modified.
`DOMNodeInserted`	`public static final String DOMNodeInserted`	Node has been added as a child of another node.

Table 2–17 (Cont.) Fields of XMLNode

Field	Syntax	Description
DOMNodeInsertedIntoDocument	public static final String DOMNodeInsertedIntoDocument	Node is being inserted into a document, either through direct insertion of the Node or insertion of a subtree in which it is contained.
DOMNodeRemoved	public static final String DOMNodeRemoved	Node is being removed from its parent node.
DOMNodeRemovedFromDocument	public static final String DOMNodeRemovedFromDocument	Node is being removed from a document, either through direct removal of the Node or removal of a subtree in which it is contained.
DOMSubtreeModified	public static final String DOMSubtreeModified	General event for notification of all changes to the document. Can be used instead of the more specific events.
ELEMENTDECL	public static final short ELEMENTDECL	An element declaration.
noncapturing	public static final String noncapturing	Handled by the event's target without being handled by one of the event's target's ancestors first.
RANGE_DELETE_EVENT	public static final String RANGE_DELETE_EVENT	Flag to delete range event.
RANGE_DELETETEXT_EVENT	public static final String RANGE_DELETETEXT_EVENT	Flag to set range delete text event
RANGE_INSERT_EVENT	public static final String RANGE_INSERT_EVENT	Flag to set range event
RANGE_INSERTTEXT_EVENT	public static final String RANGE_INSERTTEXT_EVENT	Flag to set range insert text event
RANGE_REPLACE_EVENT	public static final String RANGE_REPLACE_EVENT	Flag to replace range event
RANGE_SETTEXT_EVENT	public static final String RANGE_SETTEXT_EVENT	Flag to set range text event

Table 2–17 (Cont.) Fields of XMLNode

Field	Syntax	Description
TRAVERSAL_DELETE_EVENT	public static final String TRAVERSAL_DELETE_EVENT	Flag to set traversal delete event
TRAVERSAL_REPLACE_EVENT	public static final String TRAVERSAL_REPLACE_EVENT	Flag to set traversal replace event
XMLDECL_NODE	public static final short XMLDECL_NODE	A attribute declaration node

Table 2–18 Summary of Methods of XMLNode

Method	Description
XMLNode() on page 2-96	Constructs a new XMLNode.
addEventListener() on page 2-96	Registers event listeners on the event target (node).
appendChild() on page 2-96	Adds the node newChild to the end of the list of children of this node, and returns the new node.
cloneNode() on page 2-97	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
dispatchEvent() on page 2-97	Dispatches events into the implementations event model.
getAttributes() on page 2-98	Returns a NamedNodeMap containing the attributes of this node.
getChildNodes() on page 2-98	Returns all children of this node in a NodeList.
getColumnNumber() on page 2-98	Returns the column number debug information.
getDebugMode() on page 2-98	Returns the debug information mode.
getFirstChild() on page 2-98	Returns the first child of this node.
getLastChild() on page 2-98	Returns the last child of this node.
getLineNumber() on page 2-99	Returns the line number debug information.

Table 2–18 (Cont.) Summary of Methods of XMLNode

Method	Description
getLocalName() on page 2-99	Returns the Local Name of this node overrided by node types for which namespace is meaningful.
getNamespaceURI() on page 2-99	Returns the namespace URI of this node, overrided by node types for which namespace is meaningful.
getNextSibling() on page 2-99	Returns the node immediately following this node.
getNodeName() on page 2-99	Returns the name of the node.
getNodeType() on page 2-99	Returns the type of the node.
getNodeValue() on page 2-100	Returns the value of this node, depending on its type.
getOwnerDocument() on page 2-100	Returns the Document object associated with this node.
getParentNode() on page 2-100	Returns the parent of this node.
getPrefix() on page 2-100	Returns the prefix of this node overrided by node types for which namespace is meaningful.
getPreviousSibling() on page 2-100	Returns the node immediately preceding this node. I
getProperty() on page 2-101	Returns the value of a property of the node.
getSystemId() on page 2-101	Returns the system id of the entity containing this node.
getText() on page 2-101	Returns the non-marked-up text contained by this element.
hasAttributes() on page 2-101	Determines whether this node (if it is an element) has any attributes.
hasChildNodes() on page 2-101	Determines whether a node has any children.
insertBefore() on page 2-102	Inserts the node newChild before the existing child node refChild.
isNodeFlag() on page 2-102	Returns TRUE if the node flag information is set.
isSupported() on page 2-103	Tests whether the DOM implementation implements a specific feature and that feature is supported by this node.
print() on page 2-103	Writes the contents of this node into output.

Table 2–18 (Cont.) Summary of Methods of XMLNode

Method	Description
readExternal() on page 2-104	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
removeChild() on page 2-104	Removes the child node indicated by oldChild from the list of children.
removeEventListener() on page 2-104	Removes event listeners from the event target (node).
replaceChild() on page 2-105	Replaces the child node oldChild with newChild in the list of children, and returns the replaced node.
reportSAXEvents() on page 2-105	Reports SAX Events from a DOM Tree. Throws SAXException.
resetNodeFlag() on page 2-106	Resets the node flag information.
selectNodes() on page 2-106	Returns nodes from the tree which match the given pattern, as a NodeList.
selectSingleNode() on page 2-107	Returns the first node from the tree that matches the given pattern.
setDebugInfo() on page 2-107	Sets debug information in the node.
setNodeFlag() on page 2-108	Sets the node flag information.
setNodeValue() on page 2-108	Sets the value of this node, depending on its type.
setPrefix() on page 2-108	Sets the prefix of this node overrided by node types for which namespace is meaningful.
setProperty() on page 2-109	Sets a property of the node.
transformNode() on page 2-109	Transforms a node in the tree using the given stylesheet, and returns the resulting DocumentFragment.
valueOf() on page 2-109	Selects the value of the first node from tree that matches the pattern.
writeExternal() on page 2-110	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLNode()

Constructs a new `XMLNode`.

Syntax

```
protected  XMLNode();
```

Parameter	Description
`tag`	Name of the node.

addEventListener()

Registers event listeners on the event target (node).

Syntax

```
public void addEventListener( String type,
                              org.w3c.dom.events.EventListener listener,
                              boolean useCapture);
```

Parameter	Description
`type`	Type of event for which the listener is registered.
`listener`	The listener object.
`useCapture`	Fag to indicate if the listener wants to initiate capture.

appendChild()

Adds the node `newChild` to the end of the list of children of this node, and returns the new node. If the `newChild` is already in the tree, it is first removed. Throws `DOMException:`

- `HIERARCHY_REQUEST_ERR` raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node's ancestors.
- `WRONG_DOCUMENT_ERR` raised if `newChild` was created from a different document than the one that created this node.
- `NO_MODIFICATION_ALLOWED_ERR` raised if this node is readonly.

Syntax

```
public org.w3c.dom.Node appendChild( org.w3c.dom.Node newChild);
```

Parameter	Description
newChild	The node to add. If it is a DocumentFragment object, the entire contents of the document fragment are moved into the child list of this node.

cloneNode()

Returns a duplicate of this node; serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode returns null.). Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child Text node. Cloning any other type of node simply returns a copy of this node.

Syntax

```
public org.w3c.dom.Node cloneNode( boolean deep);
```

Parameter	Description
deep	If TRUE, recursively clone the subtree under the specified node; if FALSE, clone only the node itself (and its attributes, if it is an Element).

dispatchEvent()

Dispatches the events into the implementations event model. Throws an exception of value UNSPECIFIED_EVENT_TYPE if the Event's type was not specified by initializing the event before dispatchEvent was called.

Syntax

```
public boolean dispatchEvent(org.w3c.dom.events.Event evt);
```

Parameter	Description
evt	Indicates whether preventDefault() or stopPropogation() was called.

getAttributes()

Returns a `NamedNodeMap` containing the attributes of this node (if it is an `Element`) or `null` otherwise.

Syntax

```
public org.w3c.dom.NamedNodeMap getAttributes();
```

getChildNodes()

Returns all children of this node in a `NodeList`. If there are no children, this is a `NodeList` containing no nodes. The content of the returned `NodeList` is "live" in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the `NodeList` accessors; it is not a static snapshot of the content of the node. This is true for every `NodeList`, including the ones returned by the `getElementsByTagName` method.

Syntax

```
public org.w3c.dom.NodeList getChildNodes();
```

getColumnNumber()

Returns the column number debug information.

Syntax

```
public int getColumnNumber();
```

getDebugMode()

Returns the debug information mode.

Syntax

```
public boolean getDebugMode();
```

getFirstChild()

Returns the first child of this node. If there is no such node, this returns `null`.

Syntax

```
public org.w3c.dom.Node getFirstChild();
```

getLastChild()

Returns the last child of this node. If there is no such node, this returns `null`.

Syntax

```
public org.w3c.dom.Node getLastChild();
```

getLineNumber()

Returns the line number debug information.

Syntax

```
public int getLineNumber();
```

getLocalName()

Returns the Local Name of this node overrided by node types for which namespace is meaningful.

Syntax

```
public String getLocalName();
```

getNamespaceURI()

Returns the namespace URI of this node, overrided by node types for which namespace is meaningful.

Syntax

```
public String getNamespaceURI();
```

getNextSibling()

Returns the node immediately following this node. If there is no such node, this returns `null`.

Syntax

```
public org.w3c.dom.Node getNextSibling();
```

getNodeName()

Returns the name of the node.

Syntax

```
public String getNodeName();
```

getNodeType()

Returns the type of the node.

Syntax

```
public short getNodeType();
```

getNodeValue()

Returns the value of this node, depending on its type. Throws `DOMException`:

- `NO_MODIFICATION_ALLOWED_ERR` raised when the node is readonly.
- `DOMSTRING_SIZE_ERR` raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

Syntax

```
public String getNodeValue();
```

getOwnerDocument()

Returns the `Document` object associated with this node. This is also the `Document` object used to create new nodes. When this node is a `Document` this is `null`.

Syntax

```
public org.w3c.dom.Document getOwnerDocument();
```

getParentNode()

Returns the parent of this node. All nodes, except `Document`, `DocumentFragment`, and `Attr` may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is `null`.

Syntax

```
public org.w3c.dom.Node getParentNode();
```

getPrefix()

Returns the prefix of this node overrided by node types for which namespace is meaningful.

Syntax

```
public String getPrefix();
```

getPreviousSibling()

Returns the node immediately preceding this node. If there is no such node, this returns `null`.

Syntax

```
public org.w3c.dom.Node getPreviousSibling();
```

getProperty()

Returns the value of a property of the node.

Syntax

```
public Object getProperty( String propName);
```

Parameter	Description
propName	Name of the property.

getSystemId()

Returns the system id of the entity containing this node.

Syntax

```
public String getSystemId();
```

getText()

Returns the non-marked-up text contained by this element. For text elements, this is the raw data. For elements with child nodes, this method traverses the entire subtree and appends the text for each terminal text element, effectively stripping out the XML markup for the subtree.

Syntax

```
public String getText();
```

hasAttributes()

Determines whether this node (if it is an element) has any attributes. Returns TRUE if this node has any attributes, FALSE otherwise.

Syntax

```
public boolean hasAttributes();
```

hasChildNodes()

Determines whether a node has any children. Returns TRUE if the node has any children, FALSE if the node has no children.

Syntax

```
public boolean hasChildNodes();
```

insertBefore()

Inserts the node `newChild` before the existing child node `refChild`, and returns this node. If `refChild` is `null`, insert `newChild` at the end of the list of children. If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If the `newChild` is already in the tree, it is first removed. Throws `DOMException`:

- `HIERARCHY_REQUEST_ERR` raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node's ancestors.
- `WRONG_DOCUMENT_ERR` raised if `newChild` was created from a different document than the one that created this node.
- `NO_MODIFICATION_ALLOWED_ERR` raised if this node is readonly.
- `NOT_FOUND_ERR` raised if `refChild` is not a child of this node.

Syntax

```
public org.w3c.dom.Node insertBefore( org.w3c.dom.Node newChild,
                                      org.w3c.dom.Node refChild);
```

Parameter	Description
`newChild`	The node to insert.
`refChild`	The reference node, the node before which the new node must be inserted.

isNodeFlag()

Returns `TRUE` if the node flag information is set.

Syntax

```
public boolean isNodeFlag( int flag);
```

Parameter	Description
`flag`	The flag.

isSupported()

Tests whether the DOM implementation implements a specific feature and that feature is supported by this node. Returns `TRUE` if the feature is supported, `FALSE` otherwise.

Syntax

```
public boolean isSupported( String feature, String version);
```

Parameter	Description
`feature`	The feature being tested.
`version`	The version of the feature being tested.

print()

Writes the contents of this node into output. Throws `IOException`. The options are described in the following table.

Syntax	Description
`public void print( java.io.OutputStream out);`	Writes the contents of this node to the given output stream.
`public void print( java.io.OutputStream out, String enc);`	Writes the contents of this node to the given encoded output stream.
`public void print( java.io.PrintWriter out);`	Writes the contents of this node using the given print writer.

Parameter	Description
`out`	The output.
`enc`	Encoding to use for the output.

readExternal()

Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:

- `IOException` when there is an error in reading the input stream.
- `ClassNotFoundException` when the class is not found/

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
`in`	The ObjectInput stream used for reading the compressed stream.

removeChild()

Removes the child node indicated by `oldChild` from the list of children, and returns it. Throws `DOMException`:

- `NO_MODIFICATION_ALLOWED_ERR` raised if this node is readonly.
- `NOT_FOUND_ERR` raised if `oldChild` is not a child of this node.

Syntax

```
public org.w3c.dom.Node removeChild( org.w3c.dom.Node oldChild);
```

Parameter	Description
`oldChild`	The node being removed.

removeEventListener()

Removes event listeners from the event target (node).

Syntax

```
public void removeEventListener( String type,
                                 org.w3c.dom.events.EventListener listener,
                                 boolean useCapture);
```

Parameter	Description
type	Type of event for which the listener is registered.
listener	The listener object.
useCapture	The flag to indicate if the listener wants to initiate capture.

replaceChild()

Replaces the child node oldChild with newChild in the list of children, and returns the replaced node. If the newChild is already in the tree, it is first removed. Throws DOMException:

- HIERARCHY_REQUEST_ERR raised if this node is of a type that does not allow children of the type of the newChild node, or it the node to put in is one of this node's ancestors.
- WRONG_DOCUMENT_ERR raised if newChild was created from a different document than the one that created this node.
- NO_MODIFICATION_ALLOWED_ERR raised if this node is readonly.
- NOT_FOUND_ERR raised if oldChild is not a child of this node.

Syntax

```
public org.w3c.dom.Node replaceChild( org.w3c.dom.Node newChild,
                                      org.w3c.dom.Node oldChild);
```

Parameter	Description
newChild	The new node to put in the child list.
oldChild	The node being replaced in the list.

reportSAXEvents()

Report SAX Events from a DOM Tree. Throws SAXException.

Syntax

```
public void reportSAXEvents( org.xml.sax.ContentHandler cntHandler);
```

Parameter	Description
cntHandler	The content handler.

resetNodeFlag()

Resets the node flag information.

Syntax

```
public void resetNodeFlag( int flag);
```

Parameter	Description
flag	The node flag.

selectNodes()

Returns nodes from the tree which match the given pattern, as a NodeList. This method assumes that the pattern does not contain namespace prefixes. Throws XSLException if there is an error while doing the match. The options are described in the following table.

Syntax	Description
public org.w3c.dom.NodeList selectNodes(String pattern);	Matches using the pattern.
public org.w3c.dom.NodeList selectNodes(String pattern, NSResolver nsr);	Matches using the pattern and the namespace resolver.

Parameter	Description
pattern	XSL pattern to match.
nsr	NSResolver to resolve any prefixes that occur in given pattern.

selectSingleNode()

Returns the first node from the tree that matches the given pattern. Throws `XSLException` if there is an error while doing the match. The options are described in the following table.

Syntax	Description
`public org.w3c.dom.Node selectSingleNode( String pattern);`	Matches using the pattern.
`public org.w3c.dom.Node selectSingleNode( String pattern, NSResolver nsr);`	Matches using the pattern and the namespace resolver.

Parameter	Description
`pattern`	XSL pattern to match.
`nsr`	NSResolver to resolve any prefixes that occur in given pattern.

setDebugInfo()

Sets debug information in the node.

Syntax

```
public void setDebugInfo( int line,
                          int col,
                          String sysid);
```

Parameter	Description
`line`	The line number.
`col`	The column number.
`sysid`	The system id.

setNodeFlag()

Sets the node flag information.

Syntax

```
public void setNodeFlag( int flag);
```

Parameter	Description
flag	The node flag.

setNodeValue()

Sets the value of this node, depending on its type. Throws DOMException:

- NO_MODIFICATION_ALLOWED_ERR raised when the node is readonly.
- DOMSTRING_SIZE_ERR raised when it would return more characters than fit in a DOMString variable on the implementation platform.

Syntax

```
public void setNodeValue( String nodeValue);
```

Parameter	Description
nodeValue	The node value to set.

setPrefix()

Sets the prefix of this node overrided by node types for which namespace is meaningful. Throws DOMException.

Syntax

```
public void setPrefix( String prefix);
```

Parameter	Description
prefix	The prefix to set.

setProperty()

Sets a property of the node.

Syntax

```
public void setProperty( String propName,
                         Object propValue);
```

Parameter	Description
`propName`	Name of the property.
`propValue`	Value of the property.

transformNode()

Transforms a node in the tree using the given stylesheet, and returns the resulting DocumentFragment. Throws `XSLException`.

Syntax

```
public org.w3c.dom.DocumentFragment transformNode( XSLStylesheet xsl);
```

Parameter	Description
`xsl`	`The XSLStylesheet` to be used for transformation.

valueOf()

Selects the value of the first node from tree that matches the pattern. The options are described in the following table. Throws `XSLException` if there is an error while doing the match.

Syntax	Description
`public String valueOf( String pattern);`	Matches using the pattern.

Syntax	Description
`public String valueOf( String pattern, NSResolver nsr);`	Matches using the pattern and namespace resolver.

Parameter	Description
`pattern`	XSL pattern to match
`nsr`	NSResolver to resolve any prefixes that occur in given pattern

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws `IOException`.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
`out`	The ObjectOutput stream used to write the serialized/ compressed stream.

XMLNotation Class

This class implements the DOM `Notation` interface and represents a notation declared in the Document Type Definition.

Syntax

```
public class XMLNotation implements java.io.Externalizable
```

Table 2–19 Summary of Methods of XMLNotation

Method	Description
XMLNotation() on page 2-111	Default constructor.
cloneNode() on page 2-112	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
getNodeName() on page 2-112	Returns the name of the Notation.
getNodeType() on page 2-112	Returns a code representing the type of the underlying object.
getPublicId() on page 2-112	Returns the Public identifier; if not specified, then null.
getSystemId() on page 2-112	Returns the System identifier; if not specified, then null.
readExternal() on page 2-113	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
setPublicId() on page 2-113	Sets the Public Identifier.
setSystemId() on page 2-113	Sets the System Identifier.
writeExternal() on page 2-113	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLNotation()

Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node. In order to deserialize this node to construct the DOM node from the serialized/ compressed stream, it is required to create a handle of the object. For all normal XMLElement creation use `XMLNotation()`.

Syntax

```
public  XMLNotation();
```

cloneNode()

Returns a duplicate of this node; serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns `null`). Cloning an `Element` copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child `Text` node. Cloning any other type of node simply returns a copy of this node.

Syntax

```
public org.w3c.dom.Node cloneNode( boolean deep);
```

Parameter	Description
`deep`	If `TRUE`, recursively clones the subtree under the specified node; if `FALSE`, clones only the node itself and its attributes

getNodeName()

Returns the name of the Notation

Syntax

```
public String getNodeName();
```

getNodeType()

Returns a code representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

getPublicId()

Returns the Public identifier; if not specified, then `null` .

Syntax

```
public String getPublicId();
```

getSystemId()

Returns the System identifier; if not specified, then `null` .

Syntax

```
public String getSystemId();
```

readExternal()

Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:

- `IOException` when there is an error in reading the input stream.
- `ClassNotFoundException` when the class is not found.

Syntax

```
public void readExternal( java.io.ObjectInput inArg);
```

Parameter	Description
`in`	The ObjectInput stream used for reading the compressed stream.

setPublicId()

Sets the Public Identifier.

Syntax

```
public void setPublicId( String pubid);
```

Parameter	Description
`pubid`	Public Identifier to set.

setSystemId()

Sets the System Identifier.

Syntax

```
public void setSystemId( String url);
```

Parameter	Description
`url`	System identifier to set.

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws `IOException`.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
`out`	The ObjectOutput stream used to write the serialized/ compressed stream.

XMLNSNode Class

Extends XMLNode to add support for Namespace names and children

Syntax

```
public class XMLNSNode extends oracle.xml.parser.v2.XMLNode implements
java.lang.Cloneable java.io.Externalizable,
```

Table 2–20 Summary of Methods XMLNSNode

Method	Description
XMLNSNode() on page 2-116	Constructs a new XMLNSNode.
addText() on page 2-116	Adds text to this node, or appends it to the last child if the last child is a text node.
appendChild() on page 2-117	Adds the node newChild to the end of the list of children of this node.
getChildNodes() on page 2-117	Returns all children of this node as a NodeList.
getFirstChild() on page 2-118	Returns the first child of this node.
getLastChild() on page 2-118	Returns the last child of this node.
getLocalName() on page 2-118	Returns the Local Name of this node overrided by node types for which namespace is meaningful.
getNamespaceURI() on page 2-118	Returns the namespace URI of this node overrided by node types for which namespace is meaningful.
getNodeName() on page 2-118	Returns the name of this node, depending on its type.
getPrefix() on page 2-118	Returns the prefix of this node overrided by node types for which namespace is meaningful.
getText() on page 2-119	Returns the non-marked-up text contained by this element.
hasChildNodes() on page 2-119	Determines whether a node has any children.
insertBefore() on page 2-119	Inserts the child node before an existing child node.
normalize() on page 2-120	Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into “normal” form where only structure separates Text nodes.

Table 2–20 (Cont.) Summary of Methods XMLNSNode

Method	Description
removeChild() on page 2-120	Removes the child node indicated by oldChild from the list of children.
replaceChild() on page 2-120	Replaces the child node oldChild with newChild in the list of children.
setPrefix() on page 2-121	Sets the prefix of this node overrided by node types for which namespace is meaningful.

XMLNSNode()

Constructs a new `XMLNSNode`.

Syntax

```
protected  XMLNSNode( String tag);
```

Parameter	Description
`tag`	Name of the node.

addText()

Adds text to this node, or appends it to the last child if the last child is a text node. Throws `XMLDOMException` if text can't be added to this node. The options are described in the following table.

Syntax	Description
`public void addText( char[] ch, int start, int length);`	Adds text from a Char array.
`public XMLNode addText( String str);`	Adds text from a String.

Parameter	Description
ch	Char array to add.
start	Start index in the char array.
length	Number of chars to be added.
str	Text to add.

appendChild()

Adds the node newChild to the end of the list of children of this node, and returns that node. If the newChild is already in the tree, it is first removed. Throws DOMException:

- HIERARCHY_REQUEST_ERR raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to append is one of this node's ancestors.
- WRONG_DOCUMENT_ERR raised if newChild was created from a different document than the one that created this node.
- NO_MODIFICATION_ALLOWED_ERR raised if this node is readonly.

Syntax

```
public org.w3c.dom.Node appendChild( org.w3c.dom.Node newChild);
```

Parameter	Description
newChild	The node to add. If it is a DocumentFragment object, the entire contents of the document fragment are moved into the child list of this node.

getChildNodes()

Returns all children of this node as a NodeList. If there are no children, this is a NodeList containing no nodes. The content of the returned NodeList is "live" in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the NodeList accessors; it is not a static snapshot of the content of the node. This is true for every NodeList, including the ones returned by the getElementsByTagName method.

Syntax

```
public org.w3c.dom.NodeList getChildNodes();
```

getFirstChild()

Returns the first child of this node. If there is no such node, this returns `null`.

Syntax

```
public org.w3c.dom.Node getFirstChild();
```

getLastChild()

Returns the last child of this node. If there is no such node, this returns `null`.

Syntax

```
public org.w3c.dom.Node getLastChild();
```

getLocalName()

Returns the Local Name of this node overrided by node types for which namespace is meaningful.

Syntax

```
public String getLocalName();
```

getNamespaceURI()

Returns the namespace URI of this node. overrided by node types for which namespace is meaningful.

Syntax

```
public String getNamespaceURI();
```

getNodeName()

Returns the name of this node, depending on its type

Syntax

```
public String getNodeName();
```

getPrefix()

Returns the prefix of this node overrided by node types for which namespace is meaningful.

Syntax

```
public String getPrefix();
```

getText()

Returns the non-marked-up text contained by this element. For text elements, this is the raw data. For elements with child nodes, this method traverses the entire subtree and appends the text for each terminal text element, effectively stripping out the XML markup for the subtree. For example, if the XML document contains "William Shakespeare", `XMLDocument.getText` returns "William Shakespeare".

Syntax

```
public String getText();
```

hasChildNodes()

This is a convenience method to allow easy determination of whether a node has any children. Returns `TRUE` if the node has any children, `FALSE` otherwise.

Syntax

```
public boolean hasChildNodes();
```

insertBefore()

Inserts the node `newChild` before the existing child node `refChild`, and returns the node being inserted. If `refChild` is `null`, insert `newChild` at the end of the list of children. If `newChild` is a `DocumentFragment` object, all of its children are inserted, in the same order, before `refChild`. If the `newChild` is already in the tree, it is first removed. Throws `DOMException`:

- `HIERARCHY_REQUEST_ERR` raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node's ancestors.
- `WRONG_DOCUMENT_ERR` raised if `newChild` was created from a different document than the one that created this node. NO_MODIFICATION_ ALLOWED_ERR: Raised if this node is readonly. NOT_FOUND_ERR: Raised if `refChild` is not a child of this node.

Syntax

```
public org.w3c.dom.Node insertBefore( org.w3c.dom.Node newChild,
                                      org.w3c.dom.Node refChild);
```

Parameter	Description
`newChild`	The new node to put in the child list.

Parameter	Description
refChild	The reference node, or the node before which the new node must be inserted.

normalize()

Puts all Text nodes in the full depth of the sub-tree underneath this Node, including attribute nodes, into "normal" form where only structure (for example, elements, comments, processing instructions, CDATA sections, and entity references) separates Text nodes; there are neither adjacent Text nodes nor empty Text nodes. This can be used to ensure that the DOM view of a document is the same as if it were saved and re-loaded, and is useful when operations (such as XPointer lookups) that depend on a particular document tree structure are to be used.

Syntax

```
public void normalize();
```

removeChild()

Removes the child node indicated by oldChild from the list of children, and returns it. Throws DOMException:

- NO_MODIFICATION_ALLOWED_ERR raised if this node is readonly.
- NOT_FOUND_ERR raised if oldChild is not a child of this node.

Syntax

```
public org.w3c.dom.Node removeChild( org.w3c.dom.Node oldChild);
```

Parameter	Description
oldChild	The node being removed.

replaceChild()

Replaces the child node oldChild with newChild in the list of children, and returns the oldChild node. If the newChild is already in the tree, it is first removed. Throws DOMException:

- HIERARCHY_REQUEST_ERR raised if this node is of a type that does not allow children of the type of the newChild node, or if the node to put in is one of this node's ancestors.

- WRONG_DOCUMENT_ERR raised if newChild was created from a different document than the one that created this node.
- NO_MODIFICATION_ALLOWED_ERR raised if this node is readonly.
- NOT_FOUND_ERR raised if oldChild is not a child of this node.

Syntax

```
public org.w3c.dom.Node replaceChild( org.w3c.dom.Node newChild,
                                      org.w3c.dom.Node oldChild);
```

Parameter	Description
newChild	The new node to put in the child list.
oldChild	The node being replaced in the list.

setPrefix()

Sets the prefix of this node overrided by node types for which namespace is meaningful.

Syntax

```
public void setPrefix( String prefix);
```

Parameter	Description
prefix	The prefix of the node.

XMLOutputStream Class

This class writes output stream and can handle XML encoding.

Syntax

```
public class XMLOutputStream extends java.lang.Object
```

Table 2–21 Fields of XMLOutputStream

Field	Syntax	Description
COMPACT	public static int COMPACT	No extra indentation and new lines.
DEFAULT	public static int DEFAULT	No extra indentation and new lines.
PRETTY	public static int PRETTY	Adds indentation and new lines for readability.

Table 2–22 Summary of Methods of XMLOutputStream

Method	Description
XMLOutputStream() on page 2-123	Builds an ASCII output.
addIndent() on page 2-123	Sets indenting level for output.
close() on page 2-123	Closes the output stream.
flush() on page 2-123	Flushes the output stream.
getOutputStyle() on page 2-124	Returns the current output style.
setEncoding() on page 2-124	Sets the output character encoding.
setOutputStyle() on page 2-124	Sets the Output the style.
write() on page 2-125	Outputs character according to type of the output stream.
writeChars() on page 2-125	Writes string to the output.
writeIndent() on page 2-125	Writes an indentation.
writeNewLine() on page 2-125	Writes a new line.
writeQuotedString() on page 2-126	Writes string with surrounding quotes.

XMLOutputStream()

Builds an ASCII output. The options are described in the following table.

Syntax	Description
`public XMLOutputStream( java.io.OutputStream out);`	Builds the output using OutputStream.
`public XMLOutputStream( java.io.PrintWriter out);`	Builds the output using PrintWriter.

Parameter	Description
`out`	The output.

addIndent()

Sets indenting level for output.

Syntax

```
public void addIndent( int offset);
```

Parameter	Description
`offset`	The indenting level.

close()

Closes the output stream. Throws `IOException` if there is any error.

Syntax

```
public void close();
```

flush()

Flushes the output stream. Throws `IOException` if there is any error.

Syntax

```
public void flush();
```

getOutputStyle()

Returns the current output style.

Syntax

```
public int getOutputStyle();
```

setEncoding()

Sets the output character encoding. Throws `IOException` if error in setting the encoding type.

Syntax

```
public void setEncoding( String encoding,
                         boolean lendian,
                         boolean byteOrderMark);
```

Parameter	Description
`encoding`	The encoding of the stream.
`lendian`	The flag to indicate if the encoding is of type little endian.
`byteOrderMark`	The flag to indicate if byte order mark is set.

setOutputStyle()

Sets the Output the style.

Syntax

```
public void setOutputStyle( int style);
```

Parameter	Description
`s`	The output style

write()

Outputs character according to type of the output stream. Throws `IOException` if there is any error in writing the character.

Syntax

```
public void write( int c);
```

Parameter	Description
`c`	The character written.

writeChars()

Writes string to the output. Throws `IOException` if there is any error in writing the string.

Syntax

```
public void writeChars( String str);
```

Parameter	Description
`str`	The string that is written to the output stream.

writeIndent()

Writes an indentation. Throws `IOException` if there is any error in writing the string.

Syntax

```
public void writeIndent();
```

writeNewLine()

Writes a new line. Throws `IOException` if there is any error in writing the string.

Syntax

```
public void writeNewLine();
```

writeQuotedString()

Writes string with surrounding quotes. Throws `IOException` if there is any error in writing the string.

Syntax

```
public void writeQuotedString( String str);
```

Parameter	Description
`str`	The string that is written to the output stream.

XMLPI Class

This class implements the DOM Processing Instruction interface. See also `ProcessingInstruction, NodeFactory, DOMParser.setNodeFactory()`.

Syntax

```
public class XMLPI implements java.io.Externalizable
```

Table 2–23 Summary of Methods of XMLPI

Method	Description
XMLPI() on page 2-127	Creates a new instance of XMLPI.
addText() on page 2-128	Adds text string to the node, and returns the updated node.
getNodeName() on page 2-128	Returns the name of the PI Node.
getNodeType() on page 2-128	Returns the type of node of the underlying object.
getTarget() on page 2-128	Returns the target of this PI.
readExternal() on page 2-128	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
reportSAXEvents() on page 2-129	Reports SAX Events from a DOM Tree.
writeExternal() on page 2-129	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLPI()

Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node. In order to deserialize this node to construct the DOM node from the serialized/ compressed stream, it is required to create a handle of the object.

Syntax

```
public  XMLPI();
```

addText()

Adds text string to the node, and returns the updated node.

Syntax

```
public XMLNode addText( String str);
```

Parameter	Description
str	The Text string to be added.

getNodeName()

Returns the name of the PI Node.

Syntax

```
public String getNodeName();
```

getNodeType()

Returns the type of node of the underlying object

Syntax

```
public short getNodeType();
```

getTarget()

Returns the target of this PI. XML defines this as the first token following markup that begins the processing instruction.

Syntax

```
public String getTarget();
```

readExternal()

Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:

- `IOException` when there is an error in reading the input stream.
- `ClassNotFoundException` when the class is not found.

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
in	The ObjectInput stream used for reading the compressed stream.

reportSAXEvents()

Reports SAX Events from a DOM Tree. Throws SAXException.

Syntax

```
public void reportSAXEvents( org.xml.sax.ContentHandler cntHandler);
```

Parameter	Description
cntHandler	Content handler.

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws the IOException when there is an exception while writing the compressed stream.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
out	ObjectOutput stream used to write the compressed stream.

XMLPrintDriver Class

The `XMLPrintDriver` implements `PrintDriver` interface.

Syntax

```
public class XMLPrintDriver extends Object implements
oracle.xml.parser.v2.PrintDriver
```

Table 2–24 Fields of XMLPrintDriver

Field	Syntax	Description
`out`	`protected XMLOutputStream out`	XMLOutputStream object

Table 2–25 Summary of Methods of XMLPrintDriver

Method	Description
XMLPrintDriver() on page 2-131	Creates an instance of XMLPrintDriver.
close() on page 2-131	Closes the output stream or print writer
flush() on page 2-131	Flushes the output stream or print writer.
printAttribute() on page 2-132	Prints an XMLAttr node.
printAttributeNodes() on page 2-132	Calls print method for each attribute of the XMLElement.
printCDATASection() on page 2-132	Prints an XMLCDATA node.
printChildNodes() on page 2-132	Calls print method for each child of the XMLNode.
printComment() on page 2-133	Prints an `XML`Comment node.
printDoctype() on page 2-133	Prints a DTD.
printDocument() on page 2-133	Prints an `XMLDocument`.
printDocumentFragment() on page 2-134	Prints an empty XMLDocumentFragment object.
printElement() on page 2-134	Prints an `XMLElement`.
printEntityReference() on page 2-134	Prints an `XMLEntityReference` node

Table 2–25 (Cont.) Summary of Methods of XMLPrintDriver

Method	Description
printProcessingInstruction() on page 2-135	Prints an `XMLPI node`.
printTextNode() on page 2-135	Prints an `XMLText node`.
setEncoding() on page 2-135	Sets the encoding of the print driver.

XMLPrintDriver()

Creates an instance of XMLPrintDriver. The options are described in the following table.

Syntax	Description
`public XMLPrintDriver( OutputStream os);`	Creates an instance of XMLPrintDriver from an OutputStream.
`public XMLPrintDriver( PrintWriter pw);`	Creates an instance of XMLPrintDriver from a PrintWriter.

Parameter	Description
`os`	The OutputStream.
`pw`	The PrintWriter.

close()

Closes the output stream or print writer

Syntax

```
public void close();
```

flush()

Flushes the output stream or print writer.

Syntax

```
public void flush();
```

printAttribute()

Prints an XMLAttr node.

Syntax

```
public void printAttribute( XMLAttr attr);
```

Parameter	Description
attr	The XMLAttr Node.

printAttributeNodes()

Calls print method for each attribute of the XMLElement.

Syntax

```
public final void printAttributeNodes( XMLElement elem);
```

Parameter	Description
elem	The elem whose attributes are to be printed

printCDATASection()

Prints an XMLCDATA node.

Syntax

```
public void printCDATASection( XMLCDATA cdata);
```

Parameter	Description
cdata	The XMLCDATA node

printChildNodes()

Calls print method for each child of the XMLNode.

Syntax

```
public final void printChildNodes( XMLNode node);
```

Parameter	Description
node	The node whose children are to be printed.

printComment()

Prints an XMLComment node.

Syntax

```
public void printComment( XMLComment comment);
```

Parameter	Description
comment	The comment node.

printDoctype()

Prints an DTD.

Syntax

```
public void printDoctype( DTD dtd);
```

Parameter	Description
dtd	The DTD to be printed.

printDocument()

Prints an XMLDocument.

Syntax

```
public void printDocument( XMLDocument doc);
```

Parameter	Description
doc	The document to be printed

printDocumentFragment()

Prints an empty XMLDocumentFragment object.

Syntax

```
public void printDocumentFragment( XMLDocumentFragment dfrag);
```

Parameter	Description
dfrag	The document fragment to be printed.

printElement()

Prints an XMLElement.

Syntax

```
public void printElement( XMLElement elem);
```

Parameter	Description
elem	The element to be printed.

printEntityReference()

Prints an XMLEntityReference node

Syntax

```
public void printEntityReference( XMLEntityReference en);
```

Parameter	Description
en	The XMLEntityReference node.

printProcessingInstruction()

Prints an `XMLPI` node

Syntax

```
public void printProcessingInstruction( XMLPI pi);
```

Parameter	Description
`pi`	The XMLPI node.

printTextNode()

Prints an `XMLText` node.

Syntax

```
public void printTextNode( XMLText text);
```

Parameter	Description
`text`	The text node.

setEncoding()

Sets the encoding of the print driver.

Syntax

```
public void setEncoding( String enc);
```

Parameter	Description
`enc`	The encoding of the document being printed.

XMLRangeException Class

This class customizes the RangeException.

Syntax

```
public class XMLRangeException
```

XMLRangeException()

Generates an XMLRangeException instance.

Syntax

```
public  XMLRangeException(short code);
```

Parameter	Description
code	

XMLText Class

This class implements the DOM Text interface. See also `Text, NodeFactory, DOMParser.setNodeFactory()`.

Syntax

```
public class XMLText implements java.io.Serializable, java.io.Externalizable
```

Table 2–26 Summary of Methods of XMLText

Method	Description
XMLText() on page 2-137	Creates an instance of XMLText.
addText() on page 2-138	Adds text to the data of the text node.
getData() on page 2-138	Returns the character data of the node that implements this interface.
getNodeName() on page 2-138	Returns the name of the XMLText Node.
getNodeType() on page 2-139	Returns a type of node representing the type of the underlying object.
getNodeValue() on page 2-139	Returns String value of this text node.
isWhiteSpaceNode() on page 2-139	Checks if the text node is a whitespace node.
readExternal() on page 2-139	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
reportSAXEvents() on page 2-140	Reports SAX Events from a DOM Tree.
splitText() on page 2-140	Breaks Text node into two Text nodes at specified offset.
writeExternal() on page 2-140	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLText()

Default constructor. Note that this constructor is used only during deserialization/decompression of this DOM node. In order to deserialize this node

to construct the DOM node from the serialized/ compressed stream, it is required to create a handle of the object.

Syntax

```
public  XMLText();
```

addText()

Adds text to the data of the text node, similar to appendData.

Syntax

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

Parameter	Description
ch	char array to be appended
start	start index
length	length of the char array

getData()

Returns the character data of the node that implements this interface. The DOM implementation may not put arbitrary limits on the amount of data that may be stored in a Text node. However, implementation limits may mean that the entirety of a node's data may not fit into a single DOMString. In such cases, the user may call substringData to retrieve the data in appropriately sized pieces.

Throws DOMException:

- NO_MODIFICATION_ALLOWED_ERR raised when the node is readonly.
- DOMSTRING_SIZE_ERR raised when it would return more characters than fit in a DOMString variable on the implementation platform.

Syntax

```
public String getData();
```

getNodeName()

Returns the name of the XMLText Node.

Syntax

```
public String getNodeName();
```

getNodeType()

Returns a type of node representing the type of the underlying object.

Syntax

```
public short getNodeType();
```

getNodeValue()

Returns String value of this text node. Throws `DOMException` if any error occurs when retrieving the value.

Syntax

```
public String getNodeValue();
```

isWhiteSpaceNode()

Checks if the text node is a whitespace node.

Syntax

```
public boolean isWhiteSpaceNode();
```

readExternal()

Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. This method is called if XMLText object is deserialized (or read) as an independent node and not called from some other DOM nodeThrows the following exceptions:

- `IOException` when there is an error in reading the input stream.
- `ClassNotFoundException` -when the class is not found.

Syntax

```
public void readExternal( java.io.ObjectInput in);
```

Parameter	Description
`in`	The ObjectInput stream used for reading the compressed stream

reportSAXEvents()

Reports SAX Events from a DOM Tree. Throws `SAXException`.

Syntax

```
public void reportSAXEvents(org.xml.sax.ContentHandler cntHandler);
```

Parameter	Description
`cntHandler`	Content handler.

splitText()

Breaks `Text` node into two Text nodes at specified offset, so they are both siblings, and the node only contains content up to the offset. Returns the new Text node. New node inserted as next sibling contains all content at and after the offset point.

Throws `DOMException`:

- `INDEX_SIZE_ERR` raised if specified offset is negative or greater than number of characters in `data`.
- `NO_MODIFICATION_ALLOWED_ERR` raised if this node is readonly.

Syntax

```
public org.w3c.dom.Text splitText( int offset);
```

Parameter	Description
`offset`	Offset at which to split, starting from 0

writeExternal()

Saves the state of the object by creating a binary compressed stream with information about this object. Throws `IOException`.

Syntax

```
public void writeExternal( java.io.ObjectOutput out);
```

Parameter	Description
`out`	The ObjectOutput stream used to write the compressed stream.

3

XML Processing for Java (JAXP)

This chapter describes the JAXP APIs contained in the oracle.xml.parser.v2 package

- JXDocumentBuilder Class
- JXDocumentBuilderFactory Class
- JXSAXParser Class
- JXSAXParserFactory Class
- JXSAXTransformerFactory Class
- JXTransformer Class

See Also:

- *Oracle Application Developer's Guide - XML*

JXDocumentBuilder Class

JXDocumentBuilder defines the APIs to obtain DOM Document instances from an XML document. Using this class, an application programmer can obtain a `org.w3c.dom.Document` from XML.

An instance of this class can be obtained from the `DocumentBuilderFactory.newDocumentBuilder` method. Once an instance of this class is obtained, XML can be parsed from a variety of input sources. These input sources are InputStreams, Files, URLs, and SAX InputSources.

Note that this class reuses several classes from the SAX API. This does not require that the implementor of the underlying DOM implementation use a SAX parser to parse XML document into a `Document`. It merely requires that the implementation communicate with the application using these existing APIs.

Syntax

```
public class JXDocumentBuilder
```

Table 3–1 Summary of Methods of JXDocumentBuilder

Method	Description
getDOMImplementation() on page 3-3	Returns the associated DOM implementation object.
isNamespaceAware() on page 3-3	Indicates whether this parser understands namespaces.
isValidating() on page 3-3	Indicates whether this parser validates XML documents.
newDocument() on page 3-3	Obtains a new instance of a DOM Document object with which to build a DOM tree.
parse() on page 3-3	Parses the content of the given input source as an XML document and return a new DOM Document object.
setEntityResolver() on page 3-4	Specifies the EntityResolver to resolve entities present in the XML document to be parsed.
setErrorHandler() on page 3-4	Specifies the `ErrorHandler` to be used to resolve entities present in the XML document to be parsed.

getDOMImplementation()

Returns the associated DOM implementation object that handles this document. A DOM application may use objects from multiple implementations.

Syntax

```
public org.w3c.dom.DOMImplementation getDOMImplementation();
```

isNamespaceAware()

Indicates whether or not this parser is configured to understand namespaces.

Syntax

```
public boolean isNamespaceAware();
```

isValidating()

Indicates whether or not this parser is configured to validate XML documents.

Syntax

```
public boolean isValidating();
```

newDocument()

Obtains a new instance of a DOM Document object with which to build a DOM tree.

Syntax

```
public org.w3c.dom.Document newDocument();
```

parse()

Parses the content of the given input source as an XML document and return a new DOM Document object. Throws the following exceptions:

- `IOException` if any I/O errors occur
- `SAXException` if any parse errors occur
- `IllegalArgumentException` if the InputSource is null

Syntax

```
public org.w3c.dom.Document parse( org.xml.sax.InputSource is);
```

Parameter	Description
is	InputSource containing the content to be parsed.

setEntityResolver()

Specifies the EntityResolver to resolve entities present in the XML document to be parsed. If this is set to NULL, the underlying implementation uses its own default implementation and behavior.

Syntax

```
public void setEntityResolver( org.xml.sax.EntityResolver er);
```

Parameter	Description
er	Entity Resolver.

setErrorHandler()

Specifies the ErrorHandler to be used to resolve entities present in the XML document to be parsed. Setting this to null will result in the underlying implementation using it's own default implementation and behavior.

Syntax

```
public void setErrorHandler( org.xml.sax.ErrorHandler eh);
```

Parameter	Description
eh	Error handler.

JXDocumentBuilderFactory Class

Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.

Syntax

```
public class JXDocumentBuilderFactory
```

Table 3–2 Fields of JXDocumentBuilderFactory

Field	Syntax	Description
BASE_URL	public static final java.lang.String BASE_URL	Base URL used in parsing entities.
DEBUG_MODE	public static final java.lang.String DEBUG_MODE	Debug Mode: Boolean.TRUE or FALSE.
DTD_OBJECT	public static final java.lang.String DTD_OBJECT	DTD Object used for validation.
ERROR_ENCODING	public static final java.lang.String ERROR_ENCODING	Error encoding for error stream (only if ERROR_STREAM is set).
ERROR_STREAM	public static final java.lang.String ERROR_STREAM	Error stream; OutputStream or PrintWriter. Ignored if ErrorHandler is set.
NODE_FACTORY	public static final java.lang.String NODE_FACTORY	NodeFactory builds custom Nodes.
SCHEMA_OBJECT	public static final java.lang.String SCHEMA_OBJECT	Schema Object used for validation.
SHOW_WARNINGS	public static final java.lang.String SHOW_WARNINGS	Ignore warnings - Boolean.TRUE or FALSE.
USE_DTD_ONLY_FOR_VALIDATION	public static final java.lang.String USE_DTD_ONLY_FOR_VALIDATION	DTD Object used for validation, not added to the parser document.

Table 3–3 Summary of Methods of JXDocumentBuilderFactory

Method	Description
JXDocumentBuilderFactory() on page 3-6	Default constructor.
getAttribute() on page 3-6	Allows the user to retrieve specific attributes on the underlying implementation
isExpandEntityReferences() on page 3-7	Indicates if the factory is configured to produce parsers which expand entity reference nodes.
isIgnoringComments() on page 3-7	Indicates if the factory is configured to produce parsers which ignore comments.
isNamespaceAware() on page 3-7	Indicates if the factory is configured to produce namespace aware parsers.
newDocumentBuilder() on page 3-7	Creates a new instance of a DocumentBuilder using the currently configured parameters.
setAttribute() on page 3-7	Sets specific attributes for the implementation.

JXDocumentBuilderFactory()

Default constructor.

Syntax

```
public  JXDocumentBuilderFactory();
```

getAttribute()

Retrieves attribute values from the implementation. If the implementation doesn't recognize the attribute, throws `IllegalArgumentException`.

Syntax

```
public Object getAttribute( String name);
```

Parameter	Description
`name`	The name of the attribute.

isExpandEntityReferences()

Indicates whether or not the factory is configured to produce parsers that expand entity reference nodes. Always returns `TRUE`.

Syntax

```
public boolean isExpandEntityReferences();
```

isIgnoringComments()

Indicates whether or not the factory is configured to produce parsers which ignore comments. Always returns `FALSE`.

Syntax

```
public boolean isIgnoringComments();
```

isNamespaceAware()

Indicates whether or not the factory is configured to produce namespace aware parsers. Always returns `TRUE`.

Syntax

```
public boolean isNamespaceAware();
```

newDocumentBuilder()

Creates a new instance of a `DocumentBuilder` from currently configured parameters. Throws `ParserConfigurationException` on failure.

Syntax

```
public DocumentBuilder newDocumentBuilder();
```

setAttribute()

Sets specific attributes for the implementation. If the implementation doesn't recognize the attribute, throws `IllegalArgumentException`.

Syntax

```
public void setAttribute( String name, Object value);
```

Parameter	Description
`name`	The name of the attribute.
`value`	The value of the attribute.

JXSAXParser Class

Defines the API that wraps an `org.xml.sax.XMLReader` implementation class. In JAXP 1.0, this class wrapped the `org.xml.sax.Parser` interface, however this interface was replaced by the `XMLReader`.

For ease of transition, this class continues to support the same name and interface as well as supporting new methods. An instance of this class can be obtained from the `SAXParserFactory.newSAXParser` method. Once an instance of this class is obtained, XML can be parsed from a variety of input sources. These input sources are InputStreams, Files, URLs, and SAX InputSources.

This static method creates a new factory instance based on a system property setting or uses the platform default if no property has been defined.

The system property that controls which Factory implementation to create is named "javax.xml.style.TransformFactory". This property names a class that is a concrete subclass of this abstract class. If no property is defined, a platform default will be used. As the content is parsed by the underlying parser, methods of the given `HandlerBase` are called.

Syntax

```
public class JXSAXParser
```

Table 3–4 Summary of Methods of JXSAXParser

Method	Description
getProperty() on page 3-9	Returns the value of the requested property for in the underlying implementation of XMLReader.
getXMLReader() on page 3-9	Returns the XMLReader that is encapsulated by the implementation of this class.
isNamespaceAware() on page 3-9	Indicates if this parser is configured to understand namespaces.
isValidating() on page 3-9	Indicates if this parser is configured to validate XML documents.
setProperty() on page 3-10	Sets the particular property in the underlying implementation of XMLReader

getProperty()

Returns the value of the requested property for in the underlying implementation of org.xml.sax.XMLReader. See also `org.xml.sax.XMLReader#getProperty`. Throws the following exceptions:

- `SAXNotRecognizedException`, when the underlying XMLReader does not recognize the property name.
- `SAXNotSupportedException`, when the underlying XMLReader recognizes the property name but doesn't support the property.

Syntax

```
public java.lang.Object getProperty( String name);
```

Parameter	Description
`name`	The name of the property to be retrieved.

getXMLReader()

Returns the XMLReader that is encapsulated by the implementation of this class.

Syntax

```
public XMLReader getXMLReader();
```

isNamespaceAware()

Indicates whether or not this parser is configured to understand namespaces. Returns `TRUE` if the parser understands namespaces, `FALSE` otherwise.

Syntax

```
public boolean isNamespaceAware();
```

isValidating()

Indicates whether or not this parser is configured to validate XML documents.

Syntax

```
public boolean isValidating();
```

setProperty()

Sets the particular property in the underlying implementation of XMLReader. See also `org.xml.sax.XMLReader#setProperty`. Throws the following exceptions:

- `SAXNotRecognizedException`, when the underlying XMLReader does not recognize the property name.
- `SAXNotSupportedException`, when the underlying XMLReader recognizes the property name but doesn't support the property.

Syntax

```
public void setProperty( String name,
                         Object value);
```

Parameter	Description
`name`	The name of the property to be set.
`value`	The value of the property to be set.

JXSAXParserFactory Class

Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents.

Syntax

```
public class JXSAXParserFactory
```

Table 3–5 Summary of Methods of JXSAXParserFactory

Method	Description
JXSAXParserFactory() on page 3-11	Default constructor.
getFeature() on page 3-11	Returns the value of the requested property for in the underlying implementation of XMLReader.
isNamespaceAware() on page 3-12	Indicates if the factory is configured to produce namespace aware parsers.
newSAXParser() on page 3-12	Creates a new instance of a SAXParser using the currently configured factory parameters.
setFeature() on page 3-12	Sets the particular feature in the underlying implementation of XMLReader.

JXSAXParserFactory()

Default constructor.

Syntax

```
public  JXSAXParserFactory();
```

getFeature()

Returns the value of the requested property for in the underlying implementation of XMLReader. Throws the following exceptions:

- `SAXNotRecognizedException`, when the underlying XMLReader does not recognize the property name.
- `SAXNotSupportedException`, when the underlying XMLReader recognizes the property name but doesn't support the property.

Syntax

```
public boolean getFeature( String name);
```

Parameter	Description
name	The name of the property to be retrieved.

isNamespaceAware()

Indicates if the factory is configured to produce namespace aware parsers.

Syntax

```
public boolean isNamespaceAware();
```

newSAXParser()

Creates a new instance of a SAXParser using the currently configured factory parameters. Throws `ParserConfigurationException` if a parser which satisfies the requested configuration cannot be created.

Syntax

```
public SAXParser newSAXParser();
```

setFeature()

Sets the particular feature in the underlying implementation of XMLReader. See also `org.xml.sax.XMLReader#setFeature`. Throws the following exceptions:

- `SAXNotRecognizedException`, when the underlying XMLReader does not recognize the property name.
- `SAXNotSupportedException`, when the underlying XMLReader recognizes the property name but doesn't support the property.

Syntax

```
public void setFeature( String name, boolean value);
```

Parameter	Description
name	The name of the feature to be set.
value	The value of the feature to be set.

JXSAXTransformerFactory Class

A `JXSAXTransformerFactory` instance can be used to create Transformer and Templates objects.

The system property that determines which Factory implementation to create is named `javax.xml.transform.TransformerFactory`. This property names a concrete subclass of the `TransformerFactory`, such as `JXSAXTransformerFactory`. If the property is not defined, a platform default is be used.

This class also provides SAX-specific factory methods. It provides two types of ContentHandlers, one for creating Transformers, the other for creating Templates objects.

If an application wants to set the ErrorHandler or EntityResolver for an XMLReader used during a transformation, it should use a URIResolver to return the SAXSource which provides (with getXMLReader) a reference to the XMLReader.

Syntax

```
public class JXSAXTransformerFactory
```

Table 3–6 Summary of Methods of JXSAXTransformerFactory

Method	Description
JXSAXTransformerFactory() on page 3-14	The default constructor.
getAssociatedStylesheet() on page 3-14	Retrieves the stylesheet specification(s) associated through the xml-stylesheet processing instruction.
getAttribute() on page 3-15	Allows the user to retrieve specific attributes on the underlying implementation.
getErrorListener() on page 3-15	Retrieves the current error event handler (which should never be `NULL`) for the TransformerFactory.
getFeature() on page 3-15	Looks up the value of a feature.
getURIResolver() on page 3-16	Retrieves the object used by default during the transformation to resolve URIs.
newTemplates() on page 3-16	Processes the Source into a Templates object, which is a compiled representation of the source.

Table 3–6 (Cont.) Summary of Methods of JXSAXTransformerFactory

Method	Description
newTemplatesHandler() on page 3-16	Retrieves a TemplatesHandler object that can process SAX ContentHandler events into a Templates object.
newTransformer() on page 3-17	Generates a new Transformer object.
newTransformerHandler() on page 3-17	Generates a TransformerHandler object.
newXMLFilter() on page 3-18	Creates an XMLFilter.
setAttribute() on page 3-18	Sets specific attributes on the underlying implementation.
setErrorListener() on page 3-19	Sets the error event listener for the TransformerFactory; this is used for the processing of transformation instructions and not for the transformation itself.
setURIResolver() on page 3-19	Sets an object to use by default during the transformation to resolve URIs used in xsl:import, or xsl:include.

JXSAXTransformerFactory()

The default constructor.

Syntax

```
public  JXSAXTransformerFactory();
```

getAssociatedStylesheet()

Retrieves the stylesheet specification(s) associated through the xml-stylesheet processing instruction (see http://www.w3.org/TR/xml-stylesheet/) matching the document specified in the source parameter and other parameters. Returns a Source object suitable for passing to the TransformerFactory. Note that it is possible to return several stylesheets, in which case they are applied as if they were a list of imports or cascades in a single stylesheet.

Syntax

```
public Source getAssociatedStylesheet( Source source,
                                       String media,
                                       String title,
                                       String charset);
```

Parameter	Description
source	The XML source document.
media	The media attribute to be matched. May be NULL, in which case the preferred templates will be used (alternate = no).
title	The value of the title attribute to match. May be NULL.
charset	The value of the charset attribute to match. May be NULL.

getAttribute()

Allows the user to retrieve specific attributes on the underlying implementation. Returns the value of the attribute. Throws IllegalArgumentException if the underlying implementation doesn't recognize the attribute.

Syntax

```
public java.lang.Object getAttribute( String name);
```

Parameter	Description
name	The name of the attribute.

getErrorListener()

Retrieves the current error event handler (which should never be NULL) for the TransformerFactory.

Syntax

```
public ErrorListener getErrorListener();
```

getFeature()

Looks up the value of a feature. Returns the current state of the feature (TRUE or FALSE). The feature name is any absolute URI.

Syntax

```
public boolean getFeature( String name);
```

Parameter	Description
name	The feature name, which is an absolute URI.

getURIResolver()

Retrieves the object used by default during the transformation to resolve URIs used in document(), xsl:import(), or xsl:include(). Returns the URIResolver that was set with setURIResolver.

Syntax

```
public URIResolver getURIResolver();
```

newTemplates()

Processes the Source into a Templates object, which is a compiled representation of the source. Returns a Templates object capable of being used for transformation purposes, never NULL. This Templates object may then be used concurrently across multiple threads. Creating a Templates object allows the TransformerFactory to do detailed performance optimization of transformation instructions, without penalizing runtime transformation. If the methods fails to construct the Templates object during the parse, it throws TransformerConfigurationException.

Syntax

```
public Templates newTemplates( Source source);
```

Parameter	Description
source	An object that holds a URL, input stream, and so on.

newTemplatesHandler()

Retrieves a TemplatesHandler object that can process SAX ContentHandler events into a Templates object. Returns a non-NULL reference to a TransformerHandler, that may be used as a ContentHandler for SAX parse events. If the TemplatesHandler cannot be created, throws a TransformerConfigurationException.

Syntax

```
public TemplatesHandler newTemplatesHandler();
```

newTransformer()

Generates a new Transformer object. Returns a Transformer object that may be used to perform a transformation in a single thread, never NULL. Care must be taken not to use this object in multiple threads running concurrently; instead, different TransformerFactories should be used concurrently by different threads. Throws TransformerConfigurationException if construction of the Templates object fails during parse. The options are described in the following table.

Syntax	Description
`public Transformer newTransformer();`	Creates a new Transformer object that performs a copy of the source to the result.
`public Transformer newTransformer( Source source);`	Process the Source into a Transformer object.

Parameter	Description
source	An object that holds a URI, input stream, and so on.

newTransformerHandler()

Generates a TransformerHandler object. Returns a non-NULL reference to a TransformerHandler, ready to transform SAX parse events. The transformation is defined as an identity (or copy) transformation, for example to copy a series of SAX parse events into a DOM tree. If the TransformerHandler cannot be created, throws TransformerConfigurationException. The options are described in the following table.

Syntax	Description
`public TransformerHandler newTransformerHandler();`	Generates a TransformerHandler object that can process SAX ContentHandler events into a Result.
`public TransformerHandler newTransformerHandler( Source source);`	Uses transformation instructions specified by the source argument.

Syntax	Description
`public TransformerHandler newTransformerHandler( Templates templates);`	Uses transformation instructions specified by the `templates` argument.

Parameter	Description
`source`	The Source of the transformation instructions.
`templates`	The compiled transformation instructions.

newXMLFilter()

Creates an XMLFilter. Returns an XMLFilter object, or null if this feature is not supported. Throws `TransformerConfigurationException` if the TemplatesHandler cannot be created.

Syntax	Description
`public XMLFilter newXMLFilter( Source source);`	Creates an XMLFilter that uses the given Source as the transformation instructions.
`public XMLFilter newXMLFilter( Templates templates);`	Creates an XMLFilter that uses the given Templates as the transformation instructions.

Parameter	Description
`source`	The Source of the transformation instructions.
`templates`	The compiled transformation instructions.

setAttribute()

Sets specific attributes on the underlying implementation. An attribute in this context is defined to be an option that the implementation provides.

Syntax

```
public void setAttribute( String name,
                          Object value);
```

Parameter	Description
name	The name of the attribute.
value	The value of the attribute.

setErrorListener()

Sets the error event listener for the TransformerFactory. This is used for the processing of transformation instructions, and not for the transformation itself. Throws `IllegalArgumentException` if listener is `NULL`.

Syntax

```
public void setErrorListener(
              javax.xml.transform.ErrorListener listener);
```

Parameter	Description
listener	The new error listener.

setURIResolver()

Sets an object to use by default during the transformation to resolve URIs used in xsl:import, or xsl:include.

Syntax

```
public void setURIResolver( javax.xml.transform.URIResolver resolver);
```

Parameter	Description
name	An object that implements the URIResolver interface, or `NULL`.

JXTransformer Class

An instance of this class can transform a source tree into a result tree. An instance of this class can be obtained with the `TransformerFactory.newTransformer` method. This instance may then be used to process XML from a variety of sources and write the transformation output to a variety of sinks.

An object of this class may not be used in multiple threads running concurrently. Different Transformers may be used concurrently by different threads. A Transformer may be used multiple times. Parameters and output properties are preserved across transformations.

Syntax

```
public class JXTransformer
```

Table 3–7 Summary of Methods of JXTransformer

Method	Description
JXTransformer() on page 3-21	Constructs a JXTransformer object.
clearParameters() on page 3-21	Clears all parameters set with setParameter.
getErrorListener() on page 3-21	Retrieves the error event handler in effect for the transformation
getOutputProperties() on page 3-22	Retrieves a copy of the output properties for the transformation.
getOutputProperty() on page 3-22	Returns string value of an output property that is in effect for the transformation.
getParameter() on page 3-22	Returns a parameter that was explicitly set.
getURIResolver() on page 3-23	Returns an object that will be used to resolve URIs.
setErrorListener() on page 3-23	Sets the error event listener in effect for the transformation.
setOutputProperties() on page 3-23	Sets output properties for the transformation
setOutputProperty() on page 3-24	Sets an output property that will be in effect for the transformation.
setParameter() on page 3-25	Adds a parameter for the transformation.

Table 3–7 (Cont.) Summary of Methods of JXTransformer

Method	Description
setURIResolver() on page 3-25	Sets an object used to resolve URIs in `document()`.
transform() on page 3-26	Processes the source tree to the output result.

JXTransformer()

Constructs a `JXTransformer` object. The options are described in the following table.

Syntax	Description
`public JXTransformer();`	Default constructor.
`public JXTransformer( oracle.xml.parser.v2.XSLStylesheet templates);`	Constructs a JXTransformer object that uses the XSLStylesheet to transform the source.

Parameter	Description
`templates`	An XSLStylesheet or Templates.

clearParameters()

Clears all parameters set with setParameter.

Syntax

```
public void clearParameters();
```

getErrorListener()

Retrieves the error event handler in effect for the transformation; the error handler should never be `NULL`.

Syntax

```
public javax.xml.transform.ErrorListener getErrorListener();
```

getOutputProperties()

Retrieves a copy of the output properties for the transformation. The properties returned should contain properties set by the user and properties set by the stylesheet. These properties are "defaulted" by default properties specified by section 16 of the XSL Transformations (XSLT) W3C Recommendation. The properties that were specifically set by the user or the stylesheet should be in the base Properties list, while the XSLT default properties that were not specifically set should be the default Properties list. Thus, `getOutputProperties().getProperty()` will obtain any property in that was set by `setOutputProperty()`, `setOutputProperties()`, in the stylesheet, *or* the default properties, while `getOutputProperties().get()` will only retrieve properties that were explicitly set by `setOutputProperty()`, `setOutputProperties()`, or in the stylesheet. Note that mutation of the Properties object returned will not effect the properties that the transformation contains. If any of the argument keys are not recognized and are not namespace qualified, the property will be ignored; the behavior is not orthogonal with `setOutputProperties()`.

Syntax

```
public java.util.Properties getOutputProperties();
```

getOutputProperty()

Returns string value of an output property that is in effect for the transformation, or `NULL` if no property was found. The property specified may be a property that was set with setOutputProperty, or it may be a property specified in the stylesheet. Throws `IllegalArgumentException` if the property is not supported

Syntax

```
public String getOutputProperty( String name);
```

Parameter	Description
`name`	A non-null String that specifies an output property name, which may be namespace qualified.

getParameter()

Returns a parameter that was explicitly set with `setParameter()` or `setParameters();` `NULL` if a parameter with the given name was not found.

This method does not return a default parameter value, which cannot be determined until the node context is evaluated during the transformation process.

Syntax

```
public Object getParameter( String name);
```

Parameter	Description
name	Parameter name.

getURIResolver()

Returns an object that will be used to resolve URIs used in document(), and so on. Retrieves an object that implements the URIResolver interface, or null. Throws and `IllegalArgumentException` if listener is null.

Syntax

```
public javax.xml.transform.URIResolver getURIResolver();
```

setErrorListener()

Sets the error event listener in effect for the transformation.

Syntax

```
public void setErrorListener(javax.xml.transform.ErrorListener listener)
```

Parameter	Description
listener	The new error listener.

setOutputProperties()

Sets the output properties for the transformation. These properties will override properties set in the Templates with xsl:output. Throws an `IllegalArgumentException` if any of the argument keys are not recognized and are not namespace qualified.

If argument to this function is null, any properties previously set are removed, and the value will revert to the value defined in the templates object.

Passes a qualified property key name as a two-part string, the namespace URI enclosed in curly braces ({}), followed by the local name. If the name has a null URL, the String only contain the local name. An application can safely check for a non-null URI by testing to see if the first character of the name is a '{' character.

For example, if a URI and local name were obtained from an element defined with `<xyz:foo xmlns:xyz="http://xyz.foo.com/yada/baz.html"/>`, then the qualified name would be `"{http://xyz.foo.com/yada/baz.html}foo"`. Note that no prefix is used.

Syntax

```
public void setOutputProperties( java.util.Properties oformat);
```

Parameter	Description
`oformat`	A set of output properties that will be used to override any of the same properties in affect for the transformation.

setOutputProperty()

Sets an output property that will be in effect for the transformation. Passes a qualified property name as a two-part string, the namespace URI enclosed in curly braces ({}), followed by the local name. If the name has a null URL, the String only contain the local name. An application can safely check for a non-null URI by testing to see if the first character of the name is a '{' character.

For example, if a URI and local name were obtained from an element defined with `<xyz:foo xmlns:xyz="http://xyz.foo.com/yada/baz.html"/>`, then the qualified name would be `"{http://xyz.foo.com/yada/baz.html}foo"`. Note that no prefix is used.

The Properties object that was passed to `setOutputProperties(Properties)` won't be effected by calling this method.

Throws an `IllegalArgumentException` if the property is not supported, and is not qualified with a namespace.

Syntax

```
public void setOutputProperty(java.lang.String name, java.lang.String value)
```

Parameter	Description
name	A non-null String that specifies an output property name, which may be namespace qualified.
value	The non-null string value of the output property.

setParameter()

Adds a parameter for the transformation. Passes a qualified name as a two-part string, the namespace URI enclosed in curly braces ({}), followed by the local name. If the name has a null URL, the String only contains the local name. An application can safely check for a non-null URI by testing to see if the first character of the name is a '{' character.

For example, if a URI and local name were obtained from an element defined with `<xyz:foo xmlns:xyz="http://xyz.foo.com/yada/baz.html"/>`, then the qualified name would be `"{http://xyz.foo.com/yada/baz.html}foo"`. Note that no prefix is used.

Syntax

```
public void setParameter( String name,
                          Object value);
```

Parameter	Description
name	The name of the parameter, which may begin with a namespace URI in curly braces ({}).
value	The value object. This can be any valid Java object. It is up to the processor to provide the proper object coercion or to simply pass the object on for use in an extension.

setURIResolver()

Sets an object that will be used to resolve URIs used in `document()`. If the resolver argument is null, the URIResolver value will be cleared, and the default behavior will be used. Currently, URIResolver in document() function is not supported.

Syntax

```
public void setURIResolver( javax.xml.transform.URIResolver resolver);
```

Parameter	Description
resolver	An object that implements the URIResolver interface, or `NULL`.

transform()

Processes the source tree to the output result. Throws `TransformerException` if an unrecoverable error occurs during the course of the transformation

Syntax

```
public void transform( javax.xml.transform.Source xmlSource,
                       javax.xml.transform.Result outputTarget);
```

Parameter	Description
`xmlSource`	The input for the source tree.
`outputTarget`	The output target.

4

XSLT Processing for Java

Using the transformation specified by the XSLT stylesheet, the XSLT processor can converts an XML document into another text format.

This chapter contains these sections:

- oraxsl Class
- XPathException Class
- XSLException Class
- XSLExtensionElement Class
- XSLProcessor Class
- XSLStylesheet Class
- XSLTContext Class

See Also:

- *Oracle Application Developer's Guide - XML*

oraxsl Class

The oraxsl class provides a command-line interface to applying stylesheets on multiple XML documents. It accepts a number of command-line options that dictate how it should behave.

Syntax

```
public class oraxsl extends java.lang.Object
```

Table 4–1 Command-line options of oraxsl

command	description
`-w`	Show warnings
`-e <error log>`	A file to write errors to
`-l <xml file list>`	List of files to transform
`-d <directory>`	Directory with files to transform
`-x <source extension>`	Extensions to exclude
`-i <source extension>`	Extensions to include
`-s <stylesheet>`	Stylesheet to use
`-r <result extension>`	Extension to use for results
`-o <result extension>`	Directory to place results
`-p <param list>`	List of Params
`-t <# of threads>`	Number of threads to use
`-v`	Verbose mode

Methods of oraxsl

Table 4–2 Summary of Methods of oraxsl

Method	Description
oraxsl() on page 4-3	Class constructor.
main() on page 4-3	Invokes the oraxsl driver

oraxsl()

Class constructor.

Syntax

```
public  oraxsl();
```

main()

Invokes the oraxsl driver.

Syntax

```
public static void main( String[] args);
```

Parameter	Description
args	Command line arguments.

XPathException Class

Indicates that an exception occurred during XPath processing.

Syntax

```
public class XPathException extends oracle.xml.parser.v2.XSLException
```

Table 4–3 Summary of Methods of XPathException

Method	Description
getErrorID() on page 4-4	Retrieves error id.
getMessage() on page 4-4	Retrieves error message.

getErrorID()

Retrieves error id.

Syntax

```
public int getErrorID();
```

getMessage()

Retrieves error message. The options are described in the following table.

Syntax	Description
`public String getMessage();`	Constructs error message from error id and error params. Overrides `getMessage()` in class `java.lang.Throwable`.
`public String getMessage( XMLError err);`	Gets localized message based on the XMLError sent as parameter.

Parameter	Description
`err`	XMLError class used to get the error message.

XSLException Class

Indicates that an exception occurred during XSL transformation.

Syntax

```
public class XSLException extends oracle.xml.util.XMLException
```

XSLException()

Generates a new XSL exception.

Syntax

```
public  XSLException( String mesg);
```

Parameter	Description
mesg	Original message

XSLExtensionElement Class

Base element for extension elements

Syntax

```
public class XSLExtensionElement
```

Table 4–4 Summary of Methods of XSLExtensionElement

Method	Description
XSLExtensionElement() on page 4-6	Default constructor.
getAttributeTemplateValue() on page 4-6	Retrieves an attribute value as template.
getAttributeValue() on page 4-7	Retrieves an attribute value.
getChildNodes() on page 4-7	Retrieves the childNodes of the extension elements.
processAction() on page 4-7	Executes the body of the extension elements.
processContent() on page 4-8	Processes contents of the extension element.

XSLExtensionElement()

Default constructor.

Syntax

```
public  XSLExtensionElement();
```

getAttributeTemplateValue()

Retrieves an attribute value as template.

Syntax

```
protected final String getAttributeTemplateValue(
          XSLTContext context,
          String namespace,
          String name);
```

Parameter	Description
context	XSLT Context.
namespace	Namespace of the attribute.
name	Name of the attribute.

getAttributeValue()

Retrieves the value of an attribute.

Syntax

```
protected final String getAttributeValue( String namespace,
                                          String name);
```

Parameter	Description
namespace	Namespace of the attribute
name	Name of the attribute

getChildNodes()

Retrieves the childNodes of the extension elements as a nodelist.

Syntax

```
protected final java.util.Vector getChildNodes();
```

processAction()

Executes the body of the extension elements.

Syntax

```
public void processAction( XSLTContext context);
```

Parameter	Description
context	XSLT Context.

processContent()

Processes contents of the extension element.

Syntax

```
protected final void processContent(XSLTContext context);
```

Parameter	Description
`context`	XSLTContext.

XSLProcessor Class

This class provides methods to transform an input XML document using a previously constructed `XSLStylesheet`. The transformation effected is as specified by the XSLT 1.0 specification.

Syntax

```
public class XSLProcessor
```

Table 4–5 Summary of Methods of XSLProcessor

Method	Description
XSLProcessor() on page 4-9	Default constructor.
getParam() on page 4-9	Retrieves the value of top-level stylesheet parameter.
newXSLStylesheet() on page 4-10	Constructs an XSLStylesheet.
processXSL() on page 4-10	Transforms input XML document.
removeParam() on page 4-13	Removes the value of a top-level stylesheet parameter.
resetParams() on page 4-14	Resets all the params set.
setBaseURL() on page 4-14	Sets base url to resolve include/import hrefs.
setEntityResolver() on page 4-14	Sets entity resolver to resolve include/import hrefs.
setErrorStream() on page 4-15	Creates an output stream for the output of warnings.
setLocale() on page 4-15	Sets the locale for error reporting.
setParam() on page 4-15	Sets the value of a top-level stylesheet parameter.
showWarnings() on page 4-16	Sets the overriding XSLOutput object.

XSLProcessor()

Default constructor.

Syntax

```
public  XSLProcessor();
```

getParam()

Retrieves the value of top-level stylesheet parameter.

Syntax

```
public Object getParam( String uri,
                        String name);
```

Parameter	Description
uri	Namespace URI of the parameter.
name	Local name of the parameter.

newXSLStylesheet()

Constructs and returns a new XSLStylesheet. Throws an `XSLException`. The options are described in the following table.

Syntax	Description
`public XSLStylesheet newXSLStylesheet( InputStream xsl);`	Constructs an XSLStylesheet using the given Inputstream XSL.
`public XSLStylesheet newXSLStylesheet( Reader xsl);`	Constructs an XSLStylesheet using the given Reader.
`public XSLStylesheet newXSLStylesheet( java.net.URL xsl);`	Constructs an XSLStylesheet using the given URL.
`public XSLStylesheet newXSLStylesheet( XMLDocument xsl);`	Constructs an XSLStylesheet using the given XMLDocument.

Parameter	Description
xsl	XSL input.

processXSL()

Transforms the input XML document. Throws an `XSLException` on error. The options are described in the following table.

Syntax	Description

`public XMLDocumentFragment processXSL(`
`  XSLStylesheet xsl,`
`  InputStream xml, URL ref);`
	Transforms input XML document using given InputStream and stylesheet. Returns an XMLDocumentFragment.

`public XMLDocumentFragment processXSL(`
`  XSLStylesheet xsl,`
`  Reader xml,`
`  URL ref);`
	Transforms input XML document using given Reader and stylesheet. Returns an XMLDocumentFragment.

`public XMLDocumentFragment processXSL(`
`  XSLStylesheet xsl,`
`  URL xml,`
`  URL ref);`
	Transforms input XML document using given URL and stylesheet. Returns an XMLDocumentFragment.

`public XMLDocumentFragment processXSL(`
`  XSLStylesheet xsl,`
`  XMLDocument xml);`
	Transforms input XML document using given XMLDocument and stylesheet. Returns an XMLDocumentFragment.

`public void processXSL(`
`  XSLStylesheet xsl,`
`  XMLDocument xml,`
`  org.xml.sax.ContentHandler handler);`
	Transforms input XML document using given XMLDocument, stylesheet and content handler.

`public XMLDocumentFragment processXSL(`
`  XSLStylesheet xsl,`
`  XMLDocumentFragment xml);`
	Transforms input XML document using given XMLDocumentFragment and stylesheet. Returns an XMLDocumentFragment.

`public void processXSL(`
`  XSLStylesheet xsl,`
`  XMLDocumentFragment xml,`
`  OutputStream os);`
	Transforms input XML using given XMLDocumentFragment, stylesheet and output stream.

`public void processXSL(`
`  XSLStylesheet xsl,`
`  XMLDocumentFragment xml,`
`  printWriter pw);`
	Transforms input XML using given XMLDocumentFragment, stylesheet and print writer.

Syntax	Description

`public void processXSL(`
`  XSLStylesheet xsl,`
`  XMLDocumentFragment xml,`
`  XMLDocumentHandler handlerXML);`
	Transforms input XML document using given XMLDocumentFragment, stylesheet and XML Document handler. As the result of XSLT is a document fragment, the following functions in XMLDocumentHandler will not be called: - setDocumentLocator, startDocument, endDocument, - setDoctype, endDoctype, setXMLDecl, setTextDecl

`public void processXSL(`
`  XSLStylesheet xsl,`
`  XMLDocument xml,`
`  OutputStream out);`
	Transforms input XML document using given XMLDocument, stylesheet and output stream.

`public void processXSL(`
`  XSLStylesheet xsl,`
`  XMLDocument xml,`
`  java.io.PrintWriter pw);`
	Transforms input XML document using given XMLDocument, stylesheet and print writer.

`public void processXSL(`
`  XSLStylesheet xsl,`
`  XMLDocument xml,`
`  XMLDocumentHandler handlerXML);`
	Transforms input XML document using given XMLDocument, stylesheet and XML document handler. The output of the transformation is reported through XMLDocumentHandler. As the result of XSLT is a document fragment, the following functions in XMLDocumentHandler will not be called: - setDocumentLocator, startDocument, endDocument, - setDoctype, endDoctype, setXMLDecl, setTextDecl

`public XMLDocumentFragment processXSL(`
`  XSLStylesheet xsl,`
`  XMLElement inp);`
	Transforms input XML document using given XMLElement and stylesheet. Returns an XMLDocumentFragment.

`public void processXSL(`
`  XSLStylesheet xsl,`
`  XMLElement inp,`
`  org.xml.sax.ContentHandler handler);`
	Transforms input XML document using given XMLElement, stylesheet and content handler. The output of the transformation is reported through ContentHandler. As the result of XSLT is a document fragment, the following functions in ContentHandler will not be called: - setDocumentLocator, startDocument, endDocument,

Syntax	Description
`public void processXSL( XSLStylesheet xsl, XMLElement xml, OutputStream out);`	Transforms input XML using given XMLElement, stylesheet and output stream.
`public void processXSL( XSLStylesheet xsl, XMLElement xml, PrintWriter pw);`	Transform input XML using given XMLElement, stylesheet and print writer.
`public void processXSL( XSLStylesheet xsl, XMLElement xml, XMLDocumentHandler handlerXML);`	Transform input XML document using given XMLElement, stylesheet and document handler. As the result of XSLT is a document fragment, the following functions in XMLDocumentHandler will not be called: - setDocumentLocator, startDocument, endDocument, - setDoctype, endDoctype, setXMLDecl, setTextDecl

Parameter	Description
`xsl`	XSLStylesheet to be used for transformation.
`xml`	XML input to be transformed.
`ref`	Reference URL to resolve external entities in input xml file.
`handler`	Content handler.
`out`	Output stream to which the result is printed.
`pw`	PrintWriter to which the result is printed.
`handlerXML`	XMLDocument handler.

removeParam()

Removes the value of a top-level stylesheet parameter. Throws an `XSLException` on error.

Syntax

```
public void removeParam( String uri,
                         String name);
```

Parameter	Description
uri	URI of parameter.
name	parameter name.

resetParams()

Resets all the params set. Throws an `XSLException` on error.

Syntax

```
public void resetParams();
```

setBaseURL()

Sets base URL to resolve include/import hrefs. EntityResolver if set is used before using the base URL. See also `setEntityResolver()`.

Syntax

```
public void setBaseURL(java.net.URL url);
```

Parameter	Description
url	Base URL to be set.

setEntityResolver()

Sets entity resolver to resolve include/import hrefs. If not set, base URL (if set) is used.

Syntax

```
public void setEntityResolver( org.xml.sax.EntityResolver eResolver);
```

Parameter	Description
eResolver	Entity resolver

setErrorStream()

Creates an output stream for the output of warnings. If an output stream for warnings is not specified, the processor will not output any warnings.

Syntax

```
public final void setErrorStream(java.io.OutputStream out);
```

Parameter	Description
out	The output stream to use for errors and warnings.

setLocale()

Applications can use this to set the locale for error reporting.

Syntax

```
public void setLocale( java.util.Locale locale);
```

Parameter	Description
locale	Locale to set.

setParam()

Sets the value of a top-level stylesheet parameter. The parameter value is expected to be a valid XPath expression (note that string literal values would therefore have to be explicitly quoted). The param functions CANNOT be used along with param functions in XSLStylesheet. If the param functions in XSLProcessor are used, any parameters set using XSLStylesheet functions will be ignored. Throws an `XSLException` on error.

Syntax

```
public void setParam( String uri,
                      String name,
                      Object value);
```

Parameter	Description
uri	URI of parameter.
name	Parameter name.
value	Parameter value; Strings are treated as XPath Expr for backward compatibility).

showWarnings()

Switch to determine whether to output warnings.

Syntax

```
public final void showWarnings( boolean flag);
```

Parameter	Description
flag	Determines if warning should be shown; default: warnings not output.

XSLStylesheet Class

The class holds XSL stylesheet information such as templates, keys, variables, and attribute sets. The same stylesheet, once constructed, can be used to transform multiple XML documents.

Syntax

```
public class XSLStylesheet
```

Table 4–6 Fields of XSLStylesheet

Field	Syntax	Description
`output`	`public oracle.xml.parser.v2.XSLOutput output`	Output

Table 4–7 Summary of Methods of XSLStylesheet

Method	Description
getDecimalFormat() on page 4-18	Returns the decimal format symbols specified in the stylesheet
getOutputEncoding() on page 4-18	Returns the value of the encoding specified in `xsl:output`
getOutputMediaType() on page 4-18	Returns the value of the media-type specified in `xsl:output`
getOutputProperties() on page 4-18	Returns the output properties specified in `xsl:output as java.util.Properties.`
newTransformer() on page 4-18	Returns a JAXP Transformer object that uses this stylesheet for transformation.
getContextNode() on page 4-19	Removes the value of a top-level stylesheet parameter.
getContextPosition() on page 4-19	Resets all the params set.
getContextPosition() on page 4-19	Sets the value of a top-level stylesheet parameter.

getDecimalFormat()

Returns the decimal format symbols specified in the stylesheet.

Syntax

```
public java.text.DecimalFormatSymbols getDecimalFormat( NSName nsname);
```

Parameter	Description
nsname	Qualified name from xsl decimal-format.

getOutputEncoding()

Returns the value of the encoding specified in `xsl:output`.

Syntax

```
public String getOutputEncoding();
```

getOutputMediaType()

Returns the value of the media-type specified in `xsl:output`.

Syntax

```
public jString getOutputMediaType();
```

getOutputProperties()

Returns the output properties specified in `xsl:output` as `java.util.Properties`.

Syntax

```
public java.util.Properties getOutputProperties();
```

newTransformer()

Returns a JAXP Transformer object that uses this stylesheet for transformation.

Syntax

```
public javax.xml.transform.Transformer newTransformer();
```

XSLTContext Class

Class for Xpath processing Context.

Syntax

```
public class XSLTContext extends java.lang.Object
```

Table 4–8 Summary of Methods of XSLTContext

Method	Description
getContextNode() on page 4-19	Returns the current context node.
getContextPosition() on page 4-19	Returns the current context node position.
getContextSize() on page 4-19	Returns the current context size.
getError() on page 4-20	Returns the XMLError instance for reporting errors.
getVariable() on page 4-20	Returns variable at the given stack offset.
reportCharacters() on page 4-20	Reports characters to the current output handler.
reportNode() on page 4-20	Reports an XMLNode to the current output handler.
setError() on page 4-21	Sets the XMLError.

getContextNode()

Returns the current context node.

Syntax

```
public XMLNode getContextNode();
```

getContextPosition()

Returns the current context node position.

Syntax

```
public int getContextPosition();
```

getContextSize()

Returns the current context size.

Syntax

```
public int getContextSize();
```

getError()

Returns the XMLError instance for reporting errors.

Syntax

```
public XMLError getError();
```

getVariable()

Returns variable at the given stack offset.

Syntax

```
public getVariable( NSName name,
                    int offset);
```

Parameter	Description
name	name of the variable
offset	offset of the variable

reportCharacters()

Reports characters to the current output handler.

Syntax

```
public void reportCharacters( String data,
                              boolean disableoutesc);
```

Parameter	Description
chars	String to be printed
disableoutesc	Boolean to disable or enable escaping of characters as defined in the W3C.org XML 1.0 specification.

reportNode()

Reports a XMLNode to the current output handler.

Syntax

```
public void reportNode( XMLNode node);
```

Parameter	Description
node	Node to be output.

setError()

Sets the XMLError.

Syntax

```
public void setError(XMLError err);
```

Parameter	Description
err	Instance of XMLError.

5

XML Schema Processing

The full functionality of XML Schema Processor for Java is contained in the `oracle.XML.parser.schema` package.

This chapter discusses the following topics:

- XMLSchema Class
- XMLSchemaNode
- XSDAttribute Class
- XSDBuilder Class
- XSDComplexType Class
- XSDConstrainingFacet Class
- XSDDataValue Class
- XSDElement Class
- XSDException
- XSDGroup Class
- XSDIdentity Class
- XSDNode Class
- XSDSimpleType Class
- XSDConstantValues Interface
- XSDValidator Class

See Also:

- *Oracle Application Developer's Guide - XML*

XMLSchema Class

This class contains a set of Schema for different target namespaces. They are used by XSDParser for instance XML documents validation and by XSDBuilder as imported schemas.

Syntax

```
public class XMLSchemaNode extends oracle.xml.parser.schema.XSDNode
```

Table 5–1 Summary of Methods of XML Schema

Constructor	Description
XMLSchema() on page 5-2	XMLSchema constructor.
getAllTargetNS() on page 5-3	Returns all the Target Name space defined in the schema.
getSchemaByTargetNS() on page 5-3	Returns schemaNode for the given namespace.
getSchemaTargetNS() on page 5-3	Returns the top level schema's target Namespace.
getXMLSchemaNodeTable() on page 5-3	Returns XMLSchemaNode table.
getXMLSchemaURLS() on page 5-3	Returns XMLSchema URLs.
printSchema() on page 5-4	Print schema information.

XMLSchema()

XMLSchema constructor. Throws XSDException. The options are described in the following table.

Syntax	Description
`public XMLSchema()`	Default constructor.
`public XMLSchema(int n);`	Constructs schema given initial size of schemanode set.

Parameter	Description
n	Initial size of schemanode set

getAllTargetNS()

Returns all the Target Name space defined in the schema.

Syntax

```
public java.lang.String[] getAllTargetNS();
```

getSchemaByTargetNS()

Returns schemaNode for the given namespace.

Syntax

```
public XMLSchemaNode getSchemaByTargetNS( String namespace);
```

Parameter	Description
namespace	Target namespace of the required schema.

getSchemaTargetNS()

Returns the top level schema's target Namespace. In case there are more than one top level schema, the last one being built is returned.

Syntax

```
public String getSchemaTargetNS();
```

getXMLSchemaNodeTable()

Returns XMLSchemaNode table as a hashtable.

Syntax

```
public java.util.Hashtable getXMLSchemaNodeTable();
```

getXMLSchemaURLS()

Returns XMLSchema URLs as an array.

Syntax

```
public java.lang.String[] getXMLSchemaURLS();
```

printSchema()

Print schema information. The options are described in the following table.

Syntax	Description
`public void printSchema();`	Prints schema information.
`public void printSchema( boolean all);`	Prints schema information, including build-ins.

Parameter	Description
`all`	Flag to indicate that all information, including build-ins, should be printed

XMLSchemaNode

This class contains sets of top level Schema components in a target namespace.

Syntax

```
public class XMLSchemaNode extends oracle.xml.parser.schema.XSDNode
```

Table 5–2 Summary of Methods of XMLSchemaNode

Method	Description
XMLSchemaNode() on page 5-5	XMLSchema constructor.
getAttributeDeclarations() on page 5-5	Returns all the top level attributes in the schema.
getComplexTypeSet() on page 5-6	Returns all the top level complex type elements in the schema.
getComplexTypeTable() on page 5-6	Returns the complex type definitions.
getElementSet() on page 5-6	Returns all the top level elements in the schema.
getSimpleTypeSet() on page 5-6	Returns all the top level simpleType elements in the schema.
getSimpleTypeTable() on page 5-6	Returns the simple type definitions.
getTargetNS() on page 5-6	Returns targetNS of the schema.
getTypeDefinitionTable() on page 5-6	Returns the type definitions.

XMLSchemaNode()

`XMLSchema` constructor.

Syntax

```
public  XMLSchemaNode();
```

getAttributeDeclarations()

Returns all the top level attributes in the schema as an array.

Syntax

```
public XSDAttribute getAttributeDeclarations();
```

getComplexTypeSet()

Returns all the top level complex type elements in the schema as an array.

Syntax

```
public XSDNode getComplexTypeSet();
```

getComplexTypeTable()

Returns the complex type definitions as a hashtable.

Syntax

```
public java.util.Hashtable getComplexTypeTable();
```

getElementSet()

Returns all the top level elements in the schema as an array.

Syntax

```
public XSDNode getElementSet();
```

getSimpleTypeSet()

Returns all the top level simpleType elements in the schema as an array.

Syntax

```
public XSDNode getSimpleTypeSet();
```

getSimpleTypeTable()

Returns the simple type definitions, in a hashtable.

Syntax

```
public java.util.Hashtable getSimpleTypeTable();
```

getTargetNS()

Returns `targetNS` of the schema. Overrides `XSDNode.getTargetNS()`.

Syntax

```
public String getTargetNS();
```

getTypeDefinitionTable()

Returns the type definitions as a hashtable.

Syntax

```
public java.util.Hashtable getTypeDefinitionTable();
```

XSDAttribute Class

This class represents Schema Attribute declaration.

Syntax

```
public class XSDAttribute extends oracle.xml.parser.schema.XSDNode
```

Table 5–3 Summary of Methods of XSDAttribute

Method	Description
getDefaultVal() on page 5-7	Returns the value of 'default' attributes for element, and the value of 'value' attributes based on 'use' attribute.
getFixedVal() on page 5-7	Returns the value of 'fixed' attribute for of element, and the value of 'value' attribute based on 'use' attribute.
getName() on page 5-8	Returns the name of the node.
getRefLocalname() on page 5-8	Returns he local name of the resolved 'ref' attribute.
getRefNamespace() on page 5-8	Returns the namespace of the resolved 'ref' attribute.
getRefState() on page 5-8	Returns refState.
getTargetNS() on page 5-8	Returns target namespace.
getType() on page 5-8	Returns the node type.
isRequired() on page 5-8	Checks if the attribute is required.

getDefaultVal()

Returns the value of 'default' attr in case of element, and the value of 'value' attr based on 'use' attribute.

Syntax

```
public String getDefaultVal();
```

getFixedVal()

Returns the default value of 'fixed' attr in case of element, and the value of 'value' attr based on 'use' attribute.

Syntax

```
public java.lang.String getFixedVal();
```

getName()

Returns the name of the node. Overrides `XSDNode.getName()` in class `XSDNode`.

Syntax

```
public String getName();
```

getRefLocalname()

Returns the refLocal name of the resolved 'ref' attribute

Syntax

```
public String getRefLocalname();
```

getRefNamespace()

Returns the RefNamespace of the resolved 'ref' attribute.

Syntax

```
public String getRefNamespace();
```

getRefState()

Returns refState value. The return value is one of the following: `TYPE_UNRESOLVED`, `TYPE_RESOLVED`, `REF_UNRESOLVED`, `REF_RESOLVED`.

Syntax

```
public int getRefState();
```

getTargetNS()

Returns target namespace. `XSDNode.getTargetNS()` in class `XSDNode`.

Syntax

```
public String getTargetNS();
```

getType()

Returns the node type which is either simpleTypeor complexType.

Syntax

```
public XSDNode getType();
```

isRequired()

Checks if the attribute is required.

Syntax

```
public boolean isRequired();
```

XSDBuilder Class

Builds an XMLSchema object from XMLSchema document. XMLSchema object is a set of objects (Infoset items) corresponding to top-level schema declarations & definitions. Schema document is 'XML' parsed and converted to a DOM tree. This schema DOM tree is 'Schema' parsed in a following order: (if any) builds a schema object and makes it visible. (if any) is replaced by corresponding DOM tree. Top-level declarations & definitions are registered as a current schema infoset items. Finally, top-level tree elements (infoset items) are 'Schema' parsed. The result XMLSchema object is a set (infoset) of objects (top-level input elements). Object's contents is a tree with nodes corresponding to low-level element/group decls/refs preceded by node/object of type SNode containing cardinality information (min/maxOccurs).

Syntax

```
public class XSDBuilder
```

Table 5–4 Summary of Methods of XSDBuilder

Method	Description
XSDBuilder() on page 5-10	Class constructor.
build() on page 5-11	Builds an XMLSchema object or document.
getObject() on page 5-12	Returns XML schema object.
setEntityResolver() on page 5-12	Sets an EntityResolver for resolving imports/include.
setError() on page 5-12	Sets XMLError object.
setLocale() on page 5-13	Sets locale for error reporting.

XSDBuilder()

XSDBuilder constructor.

Syntax

```
public XSDBuilder() throws XSDException;
```

build()

Build and returns an XMLSchema object/document. An Exception is thrown if Builder fails to build an XMLSchema object. The options are described in the following table.

Syntax	Description
`public Object build( InputStream in, URL baseurl);`	Builds an XMLSchema object from an input stream and a URL.
`public Object build( Reader r, URL baseurl);`	Builds an XMLSchema object from a reader and a URL.
`public Object build( String sysId);`	Builds an XMLSchema object from system id.
`public Object build( String ns, String sysid);`	Builds an XMLSchema object from a namespace and system id.
`public Object build( String ns, URL sysid);`	Builds an XMLSchema object from a namespace and URL.
`public Object build( URL schemaurl);`	Builds an XMLSchema object from a URL.
`public Object build( XMLDocument schemaDoc);`	Builds XMLSchema from XML document.
`public Object build( XMLDocument[] schemaDocs, URL baseurl);`	Builds XMLSchema from an array of XML documents and a URL.
`public Object build( XMLDocument doc, String fragment, String ns, URL sysid);`	Build an XMLSchema object from XML document, a fragment, namespace, and system id.
`public Object build( XMLDocument schemaDoc, URL baseurl);`	Build XMLSchema from XML document and URL.

Parameter	Description
baseurl	URL used to resolve relative refs for any import/include in document
doc	XMLdocument contain the schema element
fragment	Fragment ID of the schema element
in	Inputstream of Schema
ns	Schema target namespace used to validate targetNamespace
r	Reader of Schema
schemaDoc	XMLDocument
schemaDocs	Array of XMLDocuments
sysId	Schema location
url	URL of Schema

getObject()

Returns XML schema object.

Syntax

```
public Object getObject();
```

setEntityResolver()

Sets an EntityResolver for resolving imports/include. See also **`org.xml.sax.EintityResolver.`**

Syntax

```
public void setEntityResolver( org.xml.sax.entityResolver entResolver);
```

Parameter	Description
entResolver	EntityResolver

setError()

Sets XMLError object.

Syntax

public void setError(XMLError er);

Parameter	Description
er	XMLError object

setLocale()

Sets locale for error reporting.

Syntax

```
public void setLocale(L ocale locale);
```

Parameter	Description
locale	Locale object

XSDComplexType Class

This class represents the Schema ComplexType definition.

Syntax

```
public class XSDComplexType extends oracle.xml.parser.schema.XSDNode
```

Table 5–5 Summary of Methods of XSDComplexType

Method	Description
getAttributeDeclarations() on page 5-14	Returns attribute declarations of this complex type.
getAttributeWildcard() on page 5-15	Returns attribute wildcard of this complex type.
getBaseType() on page 5-15	Returns the base type of this complextype.
getContent() on page 5-15	Returns content of XSDComplexType.
getDerivationMethod() on page 5-15	Returns information on how was this type derived from its parent type.
getElementSet() on page 5-15	Returns an array of all the local elements inside a complexType element.
getGroup() on page 5-15	Returns the attribute group or the child & attribute group.
getRefLocalname() on page 5-16	Returns the local name of resolved 'base' attr.
getTypeGroup() on page 5-16	Returns type group of Complex Type.
init() on page 5-16	Initializes complex type.
isAbstract() on page 5-16	Returns TRUE if the Complex Type is abstract.

getAttributeDeclarations()

Returns attribute declarations of this complex type; does not include wild card array of attribute declarations.

Syntax

```
public XSDAttribute getAttributeDeclarations();
```

getAttributeWildcard()

Returns attribute wildcard of this complex type.

Syntax

```
public oracle.xml.parser.schema.XSDAny getAttributeWildcard();
```

Returns

The attribute wildcard if has one

getBaseType()

Returns the base type of this complextype.

Syntax

```
public XSDNode getBaseType();
```

getContent()

Returns content of XSDComplexType.

Syntax

```
public int getContent();
```

getDerivationMethod()

Returns information on how was this type derived from its parent type. One of `EXTENSION_DERIVATION` or `RESTRICTION_DERIVATION`.

Syntax

```
public short getDerivationMethod();
```

getElementSet()

Returns an array of all the local elements inside a complexType element.

Syntax

```
public XSDNode getElementSet();
```

getGroup()

Returns the attribute group or the child & attribute group.

Syntax

```
public XSDGroup getGroup();
```

getRefLocalname()

Returns the local name of resolved 'base' attr.

Syntax

```
public java.lang.String getRefLocalname();
```

getTypeGroup()

Returns type group of Complex Type.

Syntax

```
public XSDGroup getTypeGroup();
```

init()

Initializes complex type.

Syntax

```
public static void init();
```

isAbstract()

Returns `TRUE` if the Complex Type is abstract.

Syntax

```
public boolean isAbstract();
```

XSDConstrainingFacet Class

This class represents Schema Constraining facet for Data Type.

Syntax

```
public class XSDConstrainingFacet extends java.lang.Object implements
oracle.xml.parser.schema.XSDTypeConstants
```

Table 5–6 Summary of Methods of XSDConstrainingFacet

Method	Description
getFacetId() on page 5-17	Returns the facet id.
getLexicalEnumeration() on page 5-17	Returns the LEXICAL enumeration of the constraining facet.
getLexicalValue() on page 5-17	Returns the LEXICAL value of the constraining facet.
getName() on page 5-18	Returns the name of the constraining facet.
isFixed() on page 5-18	Checks whether facet is fixed and cannot be changed.
setFixed() on page 5-18	Sets if the facet is fixed and cannot be changed.
validateFacet() on page 5-18	Validates the value against the constraint facet.

getFacetId()

Returns the facet id.

Syntax

```
public int getFacetId();
```

getLexicalEnumeration()

Returns the `LEXICAL` enumeration of the constraining facet.

Syntax

```
public java.util.Vector getLexicalEnumeration();
```

getLexicalValue()

Returns the `LEXICAL` value of a constraining facet.

Syntax

```
public String getLexicalValue();
```

getName()

Returns the name of the constraining facet.

Syntax

```
public String getName();
```

isFixed()

Checks whether facet is fixed and cannot be changed. Returns `TRUE` for fixed, `FALSE` otherwise.

Syntax

```
public boolean isFixed();
```

setFixed()

Sets whether facet is fixed and cannot be changed. `TRUE` for fixed, `FALSE` otherwise.

Syntax

```
public void setFixed( boolean fixed);
```

Parameter	Description
`value`	Value being validated.

validateFacet()

Validates the value against the constraint facet.

Syntax

```
public void validateFacet( XSDDataValue value);
```

Parameter	Description
`value`	Value being validated.

XSDDataValue Class

This class represents data values for Schema Simple Type.

Syntax

```
public class XSDDataValue extends java.lang.Object implements
oracle.xml.parser.schema.XSDTypeConstants
```

Table 5–7 Summary of Methods of XSDDataValue

Method	Description
compareTo() on page 5-19	Compares two values returns -1 for smaller, 0 for equal, and 1 for greater.
getLength() on page 5-19	Returns the length of `STRING/BINARY` value.
getLexicalValue() on page 5-19	Returns `LEXICAL` value from the `XSDDataValue` class.
getPrecision() on page 5-20	Returns the precision of decimal value.
getScale() on page 5-20	Returns the scale of decimal value.

compareTo()

Compares two values; returns -1 for smaller, 0 for equal, and 1 for greater. Throws `XSDException` if the data values are not comparable.

Syntax

```
public int compareTo( XSDDataValue val);
```

getLength()

Gets the length of `STRING/BINARY` value. Throws `XSDException` if the data value is neither of these types.

Syntax

```
public int getLength();
```

getLexicalValue()

Returns `LEXICAL` value from the `XSDDataValue` class.

Syntax

```
public String getLexicalValue();
```

getPrecision()

Returns the precision of decimal value. Throws `XSDException` if the data values are not decimal type.

Syntax

```
public int getPrecision();
```

getScale()

Returns the int value of a decimal scale. Throws `XSDException` if the data value is not decimal type.

Syntax

```
public int getScale();
```

XSDElement Class

The `XSDElement` class represents the Schema element declaration.

Syntax

```
public class XSDElement
```

Table 5–8 Summary of Methods of XSDElement

Method	Description
findEquivClass() on page 5-22	Finds the equivalent class corresponding to this class.
getDefaultVal() on page 5-22	Returns the value of 'default' attr in case of element, and the value of 'value' attr based on 'use' attribute.
getEquivClassRef() on page 5-22	Returns the local name of the resolved equivalent class.
getFixedVal() on page 5-22	Returns the value of 'fixed' attr in case of element, and the value of 'value' attr based on 'use' attribute.
getIdentities() on page 5-22	Returns the set of identities.
getMaxOccurs() on page 5-23	Returns the maxOccurs.
getMinOccurs() on page 5-23	Returns the minOccurs.
getName() on page 5-23	Returns Name.
getRefLocalname() on page 5-23	Returns the local name of the resolved 'ref' attribute.
getRefNamespace() on page 5-23	Returns the namespace of the resolved 'ref' attribute.
getRefState() on page 5-23	Returns refState.
getSubstitutionGroup() on page 5-23	Returns the substitutionGroup.
getTargetNS() on page 5-24	Sets target namespace.
getType() on page 5-24	Sets the node type.
isAbstract() on page 5-24	Returns TRUE if element abstract.
isNullable() on page 5-24	Returns TRUE if element nullable.
setMaxOccurs() on page 5-24	Sets the maxOccurs.
setMinOccurs() on page 5-24	Sets the minOccurs.

findEquivClass()

Finds the equivalent class corresponding to this class.

Syntax

```
public XSDElement findEquivClass( String ns,
                                  String nm);
```

Parameter	Description
ns	Namespace.
rm	Name.

getDefaultVal()

Returns the value of 'default' attr in case of element, and the value of 'value' attr based on 'use' attribute.

Syntax

```
public String getDefaultVal();
```

getEquivClassRef()

Returns the local name of the resolved equivalent class.

Syntax

```
public String getEquivClassRef();
```

getFixedVal()

Returns the value of 'fixed' attr in case of element, and the value of 'value' attr based on 'use' attribute

Syntax

```
public java.lang.String getFixedVal();
```

getIdentities()

Returns the set of identities, as an array.

Syntax

```
public XSDIdentity getIdentities();
```

getMaxOccurs()

Returns the maxOccurs

Syntax

```
public int getMaxOccurs();
```

getMinOccurs()

Returns the minOccurs.

Syntax

```
public int getMinOccurs();
```

getName()

Returns the name of the node.

Syntax

```
public String getName();
```

getRefLocalname()

Returns the local name of the resolved 'ref' attribute

Syntax

```
public String getRefLocalname();
```

getRefNamespace()

Returns the namespace of the resolved 'ref' attribute

Syntax

```
public String getRefNamespace();
```

getRefState()

Returns refState. The return value is one of the following: `TYPE_UNRESOLVED`, `TYPE_RESOLVED`, `REF_UNRESOLVED`, `REF_RESOLVED`.

Syntax

```
public int getRefState();
```

getSubstitutionGroup()

Returns the substitutionGroup

Syntax

```
public java.util.Vector getSubstitutionGroup();
```

getTargetNS()

Returns target namespace.

Syntax

```
public java.lang.String getTargetNS();
```

getType()

Returns the node type, either simpleType or complexType.

Syntax

```
public XSDNode getType();
```

isAbstract()

Returns `TRUE` if this element is abstract.

Syntax

```
public boolean isAbstract();
```

isNullable()

Returns `TRUE` if this element is nullable.

Syntax

```
public boolean isNullable();
```

setMaxOccurs()

Sets the maxOccurs.

Syntax

```
public void setMaxOccurs( int max);
```

Parameter	Description
`maxOccurs`	value

setMinOccurs()

Sets the minOccurs.

Syntax

```
public void setMinOccurs( int min);
```

Parameter	Description
minOccurs	value

XSDException

Indicates that an exception occurred during XMLSchema validation.

Syntax

```
public class XSDException extends Exception
```

getMessage()

Overrides getMessage() in class Throwable in order to construct error message from error id and error parameters. The options are described in the following table.

Syntax	Description
`public String getMessage()`	Constructs error message from error id and error parameters
`public String getMessage( XMLError err)`	Constructs localized error message based on the XMLError sent as parameter

Parameter	Description
`err`	XMLError class used to get the error message

XSDGroup Class

The `XSDGroup` class represents the Schema group definition.

Syntax

```
public class XSDGroup
```

Table 5–9 Summary of Methods of XSDIdentity

Method	Description
getMaxOccurs() on page 5-27	Returns the maxOccurs.
getMinOccurs() on page 5-27	Returns the minOccurs.
getNodeVector() on page 5-27	Returns the particles of the group stored in nodeVector.
getOrder() on page 5-28	Returns the composite type - `ALL`, `SEQUENCE`, `CHOICE`.
setMaxOccurs() on page 5-28	Sets maxOccurs.
setMinOccurs() on page 5-28	Sets minOccurs.

getMaxOccurs()

Returns the maxOccurs.

Syntax

```
public int getMaxOccurs();
```

getMinOccurs()

Returns the minOccurs.

Syntax

```
public int getMinOccurs();
```

getNodeVector()

Returns the particles of the group stored in nodeVector.

Syntax

```
public java.util.Vector getNodeVector();
```

getOrder()

Returns the composite type - `ALL`, `SEQUENCE`, or `CHOICE`

Syntax

```
public int getOrder();
```

setMaxOccurs()

Sets maxOccurs.

Syntax

```
public void setMaxOccurs( int max);
```

Parameter	Description
`maxOccurs`	value

setMinOccurs()

Sets the minOccurs.

Syntax

```
public void setMinOccurs( int min);
```

Parameter	Description
`minOccurs`	value

XSDIdentity Class

Represents Schema Identity-Constraint definition: `Unique`, `Key`, and `Keyref`.

Syntax

```
public class XSDIdentity extends oracle.xml.parser.schema.XSDNode
```

Table 5–10 Summary of Methods of XSDIdentity

Method	Description
getFields() on page 5-29	Returns the fields.
getNodeType() on page 5-29	Returns the node type.
getRefer() on page 5-29	Returns the reference key.
getSelector() on page 5-29	Returns the selector.

getFields()

Returns the fields.

Syntax

```
public java.lang.String[] getFields();
```

getNodeType()

Returns the node Type. Overrides `XSDNode.getNodeType()` in class `XSDNode`.

Syntax

```
public int getNodeType();
```

getRefer()

Returns the referenced key

Syntax

```
public String getRefer();
```

getSelector()

Returns the selector.

Syntax

```
public String getSelector();
```

XSDNode Class

Root class for most of XSD classes. Contains fields and methods corresponding to XMLSchema definition attributes.

Syntax

```
public class XSDNode
```

Table 5–11 Summary of Methods of XSDNode

Method	Description
getName() on page 5-31	Returns the name of the node.
getNamespaceURI() on page 5-31	Returns namespace uri for psv.
getNodeType() on page 5-31	Returns the type of XSDNode.
getTargetNS() on page 5-32	Returns target namespace.
isNodeType() on page 5-32	Checks if the node is of the give type.

getName()

Returns the name of the node.

Syntax

```
public java.lang.String getName();
```

getNamespaceURI()

Returns namespace uri for psv.

Syntax

```
public String getNamespaceURI()
```

getNodeType()

Returns the type of XSDNode.

Syntax

```
public int getNodeType();
```

getTargetNS()

Returns target namespace.

Syntax

```
public java.lang.String getTargetNS();
```

isNodeType()

Checks if the node is of the give type.

Syntax

```
public boolean isNodeType( int type);
```

Parameter	Description
type	Type of node that is being checked.

XSDSimpleType Class

This class represents Schema Simple Type definition.

Syntax

```
public class XSDSimpleType implements oracle.xml.parser.schema.XSDTypeConstants
```

Table 5–12 Summary of Methods of XSDSimpleType

Method	Description
XSDSimpleType() on page 5-33	Class constructor.
derivedFrom() on page 5-34	Derive a type from the given base type.
getBase() on page 5-34	Returns the base type of this type.
getBasicType() on page 5-35	Gets the basic type from which this type was derived.
getBuiltInDatatypes() on page 5-35	Gets a built-in datatype.
getFacets() on page 5-35	Get the facets.
getMaxOccurs() on page 5-35	Get the value of maxOccurs.
getMinOccurs() on page 5-35	Get the value of minOccurs.
getVariety() on page 5-35	Get the variety of the type.
isAbstract() on page 5-36	Returns TRUE if this Type is abstract.
setFacet() on page 5-36	Sets a facet for the datatype (Internal private API).
setMaxOccurs() on page 5-36	Sets the value of maxOccurs.
setMinOccurs() on page 5-36	Sets the value of minOccurs.
setSource() on page 5-37	Sets the base type of the datatype; for aggregate types sets the type of the component of the aggregate type.
validateValue() on page 5-37	Validates the string value with the facets defined for this type.

XSDSimpleType()

Class constructor. The options are described in the following table.

Syntax	Description
`public XSDSimpleType();`	Default constructor.
`public XSDSimpleType( int basic, String tnm);`	Implements W3C XMLSchema Datatype definition.

Parameter	Description
`basic`	Id of the Primitive type from which this one is derived.
`tnm`	Name of this type.

derivedFrom()

Derives a type from the given base type. Throws `XSDException` if new type cannot be created.

Syntax

```
public static XSDSimpleType derivedFrom( XSDSimpleType source,
                                         String nm,
                                         String var);
```

Parameter	Description
`source`	XSDSimpleType The base type
`nm`	The name of the new type
`var`	The method of derivation

getBase()

Retrieves the base type.

Syntax

```
public XSDSimpleType getBase();
```

getBasicType()

Retrieves the basic type from which this type was derived.

Syntax

```
public int getBasicType();
```

getBuiltInDatatypes()

Gets a built-in datatype. Throws `XSDException` if the type is not a valid name.

Syntax

```
public static Hashtable getBuiltInDatatypes();
```

Parameter	Description
`type`	Name of the built-in type.

getFacets()

Retrieves the facets.

Syntax

```
public XSDConstrainingFacet getFacets();
```

getMaxOccurs()

Retrieves value of maxOccurs.

Syntax

```
public int getMaxOccurs();
```

getMinOccurs()

Retrieves the value of minOccurs.

Syntax

```
public int getMinOccurs();
```

getVariety()

Retrieves the variety of the type.

Syntax

```
public java.lang.String getVariety();
```

isAbstract()

Returns `TRUE` if this type is abstract, `FALSE` otherwise.

Syntax

```
public boolean isAbstract();
```

setFacet()

Sets a facet for the datatype (Internal private API). Throws `XSDException` if the facet is invalid.

Syntax

```
public void setFacet( String facetName,
                      String value);
```

Parameter	Description
`facetName`	Name of the facet being set.
`value`	Value of the facet.

setMaxOccurs()

Set the value of maxOccurs

Syntax

```
public void setMaxOccurs(int max);
```

Parameter	Description
`max`	Number of maximum occurrences.

setMinOccurs()

Set the value of minOccurs.

Syntax

```
public void setMinOccurs( int min);
```

Parameter	Description
min	Number of minimum occurrences.

setSource()

Sets the base type of the datatype, or in case of aggregate types sets the type of the component of the aggregate type. Throws SDException if the src is not a valid type.

Syntax

```
public void setSource( XSDNode src);
```

Parameter	Description
src	XSDNode source.

validateValue()

Validates the string value with the facets defined for this type. Throws XSDException if the value is not valid.

Syntax

```
public void validateValue( String val);
```

Parameter	Description
val	Value to be validated.

XSDConstantValues Interface

This interface defines constants for the W3C Schema Processor.

Syntax

```
public interface XSDTypeConstants
```

Table 5–13 Constants Defined in XSDConstantValues

Constant	Definition
_abstract	public static final String _abstract
_all	public static final String _all
_annotation	public static final String _annotation
_any	public static final String _any
_anyAttribute	public static final String _anyAttribute
_anySimpleType	public static final String _anySimpleType
_anyType	public static final String _anyType
_attrFormDefault	public static final String _attrFormDefault
_attribute	public static final String _attribute
_attributeGroup	public static final String _attributeGroup
_attrTag	public static final String _attrTag
_base	public static final String _base
_block	public static final String _block
_blockDefault	public static final String _blockDefault
_choice	public static final String _choice
_complexContent	public static final String _complexContent
_complexType	public static final String _complexType
_content	public static final String _content
_default	public static final String _default
_derivedBy	public static final String _derivedBy

Table 5–13 (Cont.) Constants Defined in XSDConstantValues

Constant	Definition
_element	public static final String _element
_elementOnly	public static final String _elementOnly
_elemFormDefault	public static final String _elemFormDefault
_empty	public static final String _empty
_enumeration	public static final String _enumeration
_equivClass	public static final String _equivClass
_extension	public static final String _extension
_false	public static final String _false
_field	public static final String _field
_final	public static final String _final
_finalDefault	public static final String _finalDefault
_fixed	public static final String _fixed
_form	public static final String _form
_group	public static final String _group
_id	public static final String _id
_import	public static final String _import
_include	public static final String _include
_itemType	public static final String _itemType
_key	public static final String _key
_keyref	public static final String _keyref
_lax	public static final String _lax
_list	public static final String _list
_maxOccurs	public static final String _maxOccurs
_memberTypes	public static final String _memberTypes
_minOccurs	public static final String _minOccurs
_mixed	public static final String _mixed
_nall	public static final String _nall

Table 5–13 (Cont.) Constants Defined in XSDConstantValues

Constant	Definition
_name	public static final String _name
_namespace	public static final String _namespace
_nil	public static final String _nil
_nillable	public static final String _nillable
_nnany	public static final String _nnany
_nnlist	public static final String _nnlist
_nnlocal	public static final String _nnlocal
_nnother	public static final String _nnother
_nntargetNS	public static final String _nntargetNS
_noNSSchemaLocation	public static final String _noNSSchemaLocation
_notation	public static final String _notation
_null	public static final String _null
_nullable	public static final String _nullable
_optional	public static final String _optional
_pattern	public static final String _pattern
_processContents	public static final String _processContents
_prohibited	public static final String _prohibited
_publicid	public static final String _publicid
_qualified	public static final String _qualified
_redefine	public static final String _redefine
_ref	public static final String _ref
_refer	public static final String _refer
_required	public static final String _required
_restriction	public static final String _restriction
_restrictions	public static final String _restrictions
_schema	public static final String _schema
_schemaLocation	public static final String _schemaLocation

Table 5–13 (Cont.) Constants Defined in XSDConstantValues

Constant	Definition
_selector	public static final String _selector
_sequence	public static final String _sequence
_simpleContent	public static final String _simpleContent
_simpleType	public static final String _simpleType
_skip	public static final String _skip
_strict	public static final String _strict
_substitution	public static final String _substitution
_systemid	public static final String _systemid
_targetNS	public static final String _targetNS
_textOnly	public static final String _textOnly
_this	public static final String _this
_true	public static final String _true
_type	public static final String _type
_undef	public static final String _undef
_union	public static final String _union
_unique	public static final String _unique
_unqualified	public static final String _unqualified
_use	public static final String _use
_value	public static final String _value
_version	public static final String _version
_xmlns	public static final String _xmlns
ABSENT_NS	public static final int ABSENT_NS
ACCEPTED	public static final int ACCEPTED
ALL	public static final int ALL
ANY	public static final int ANY
ANY_ATTRIBUTE	public static final int ANY_ATTRIBUTE
ANY_NODE	public static final String ANY_NODE

Table 5–13 (Cont.) Constants Defined in XSDConstantValues

Constant	Definition
ATTRIBUTE	public static final int ATTRIBUTE
ATTRIBUTE_GROUP	public static final int ATTRIBUTE_GROUP
AUTO_VALIDATION	public static final String AUTO_VALIDATION
BASE_RESOLVED	public static final int BASE_RESOLVED
BASE_UNRESOLVED	public static final int BASE_UNRESOLVED
BASE_URL	public static final String BASE_URL
CHOICE	public static final int CHOICE
constName	public static final String constName[]
cyclicChain	public static final int cyclicChain
DATATYPE	public static final int DATATYPE
DONE	public static final int DONE
ELEMENT	public static final int ELEMENT
ELEMENT_CHILD	public static final int ELEMENT_CHILD
ELEMENT_ONLY	public static final int ELEMENT_ONLY
elemNotNullable	public static final int elemNotNullable
EMPTY	public static final int EMPTY
EQUIV_RESOLVED	public static final int EQUIV_RESOLVED
EQUIV_UNRESOLVED	public static final int EQUIV_UNRESOLVED
ERROR	public static final int ERROR
EXTENTION	public static final int EXTENTION
FACET_CHILD	public static final int FACET_CHILD
FAKE_NODE	public static final XSDGroup FAKE_NODE
FIXED_SCHEMA	public static final String FIXED_SCHEMA
GROUP	public static final int GROUP
IDENTITY_KEY	public static final int IDENTITY_KEY
IDENTITY_KEYREF	public static final int IDENTITY_KEYREF
IDENTITY_UNIQUE	public static final int IDENTITY_UNIQUE

Table 5–13 (Cont.) Constants Defined in XSDConstantValues

Constant	Definition
IMPORT	public static final int IMPORT
INCLUDE	public static final int INCLUDE
INFINITY	public static final int INFINITY
invalidAttr	public static final int invalidAttr
invalidAttrVal	public static final int invalidAttrVal
invalidChars	public static final int invalidChars
invalidContent	public static final int invalidContent
invalidDerivation	public static final int invalidDerivation
invalidElem	public static final int invalidElem
invalidETag	public static final int invalidETag
invalidFacet	public static final int invalidFacet
invalidFixedChars	public static final int invalidFixedChars
invalidNameRef	public static final int invalidNameRef
invalidNS	public static final int invalidNS
invalidParsAttr	public static final int invalidParsAttr
invalidPrefix	public static final int invalidPrefix
invalidRef	public static final int invalidRef
invalidSTag	public static final int invalidSTag
invalidTargetNS	public static final int invalidTargetNS
LAX_VALIDATION	public static final String LAX_VALIDATION
missingAttr	public static final int missingAttr
MIXED	public static final int MIXED
needsSource	public static final int needsSource
NEW_STATE	public static final int NEW_STATE
NO_CHILD	public static final int NO_CHILD
noDefinition	public static final int noDefinition
NOT_DONE	public static final int NOT_DONE

Table 5–13 (Cont.) Constants Defined in XSDConstantValues

Constant	Definition
NOTATION	public static final int NOTATION
notComplete	public static final int notComplete
notSubTypeOf	public static final int notSubTypeOf
NS_FRAME	public static final int NS_FRAME
NS_RESOLVER	public static final String NS_RESOLVER
REDEFINE	public static final int REDEFINE
REF_RESOLVED	public static final int REF_RESOLVED
REF_UNRESOLVED	public static final int REF_UNRESOLVED
refToAbstractElem	public static final int refToAbstractElem
refToAbstractType	public static final int refToAbstractType
RESTRICTION	public static final int RESTRICTION
SCHEMA_NS	public static final int SCHEMA_NS
SEQ	public static final int SEQ
STRICT_VALIDATION	public static final String STRICT_VALIDATION
TEXT_ONLY	public static final int TEXT_ONLY
TOP_LEVEL	public static final int TOP_LEVEL
TYPE	public static final int TYPE
TYPE_RESOLVED	public static final int TYPE_RESOLVED
TYPE_UNRESOLVED	public static final int TYPE_UNRESOLVED
UNDEF	public static final int UNDEF
undefinedType	public static final int undefinedType
unexpectedAttr	public static final int unexpectedAttr
unexpectedElem	public static final int unexpectedElem
unnamedAttrDecl	public static final int unnamedAttrDecl
VALIDATION_MODE	public static final String VALIDATION_MODE
XSDAUG2000NS	public static final String XSDAUG2000NS
XSDDATATYPENS	public static final String XSDDATATYPENS

Table 5–13 (Cont.) Constants Defined in XSDConstantValues

Constant	Definition
XSDNAMESPACE	public static final String XSDNAMESPACE
XSDRECNS	public static final String XSDRECNS
XSDRECTYPENS	public static final String XSDRECTYPENS
XSI2000NS	public static final String XSI2000NS
XSINAMESPACE	public static final String XSINAMESPACE
XSIRECNS	public static final String XSIRECNS

XSDValidator Class

XSDValidator validates an instance XML document against an XMLSchema.

When registered, an XSDValidator object is inserted as a pipe-line node between XMLParser and XMLDocument events handler (SAXHandler or DOMBuilder). It works with three events: `startElement()`, `characters()` and `endElement()`. If defined, default element and default attribute values are added to the events contents as XMLSchema additions to infoset, and are propagated upwards.

The XMLSchema object is a set or group of element declarations:

```
[element(name)] -> [shode(min/maxOccurs)] -> [type(group/simpleType)]
```

XSDValidator is implemented as stack based state machine. Each state represents element type - `group` or `simpleType`.

XMLSchema object, as a group, is loaded as a first state. Current `element` is matched against current state group elements. If matched the element type element name and snode info are loaded as new state.

In a case of group, a vector of `counters` is allocated in a parallel stack. This vector is used to count element occurrences.

State status could be:

- `NEW_STATE`: just loaded and not tried.
- `ACCEPTED: minOccurs` satisfied. Could still accept element occurrences.
- `DONE`: `maxOccurs` satisfied. Doesn't accept element occurrences.

Text element contents, or event characters, are matched against `simpleType` through method `validateValue()`. End element, found through event `endElement()`, is matched against the last named state.

XMLSchema attributes are represented as a group (`attrName -> attrType`) forming the contents of special element: `<_attrTag> attrType`. XMLParser converts attributes, found through event `startElement()`, accordingly.

XSDAny objects are used as Namespace frame descriptors.

Fake states are loaded in a case of error or when wildcard('any') contents is skipped.

Syntax

```
public class XSDValidator
```

Table 5–14 Summary of Methods of XSDValidator

Method	Description
XSDValidator() on page 5-47	Class constructor.
characters() on page 5-47	Propagates notification of character data in elements.
endElement() on page 5-48	Receives notification of the end of an element.
setDocumentLocator() on page 5-48	Propagates Locator object for document events.
setError() on page 5-48	Sets an XMLError object as current err.
setXMLProperties() on page 5-49	Sets XML Properties for runtime properties.
setXMLProperty() on page 5-49	Sets a property.
startElement() on page 5-49	Receive notification of a beginning of the element.

XSDValidator()

XSDValidator constructor.

Syntax

```
public  XSDValidator();
```

characters()

Propagate notification of character data inside an element. Throws `org.xml.sax.SAXException`. See also `org.xml.sax.DocumentHandler`

Syntax

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

Parameter	Description
`ch`	The characters
`start`	The start position in the character array.
`length`	The number of characters to use from the character array.

endElement()

Receive notification of the end of an element. Throws `org.xml.sax.SAXException`.

Syntax

```
public void endElement( String namespaceURI,
                        String localName,
                        String qName);
```

Parameter	Description
`uri`	The Namespace URI, or the empty string if unavailable or no Namespace.
`localName`	The local name without prefix, or empty string if no Namespace processing.
`qName`	The qualified XML 1.0 name with prefix; empty string if not available.

setDocumentLocator()

Propagates Locator object for document events. See also `org.xml.sax.DocumentHandler`, `org.xml.sax.Locator`.

Syntax

```
public void setDocumentLocator( org.xml.sax.Locator locator);
```

Parameter	Description
`locator`	A locator for all SAX document events

setError()

Sets an XMLError object as current err. Throws `org.xml.sax.SAXException`.

Syntax

```
public void setError( oracle.xml.parser.v2.XMLError he);
```

Parameter	Description
`he`	XMLError object.

setXMLProperties()

Sets XML Properties for runtime properties.

Syntax

```
public void setXMLProperties( XMLProperties xmlProp);
```

Parameter	Description
xmlProp	XML properties.

setXMLProperty()

Sets property value and returns the property. A null is returned if the property is read-only and cannot be set, or is not supported.

Syntax

```
public Object setXMLProperty( java.lang.String name,
                              java.lang.Object value);
```

Parameter	Description
name	name of the property
value	value of the property

startElement()

Receives notification of the beginning of an element. Throws `org.xml.sax.SAXException`. See also: `org.xml.sax.Attributes`, `endElement()`.

Syntax

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

Parameter	Description
`uri`	The Namespace URI, or the empty string if the element has no Namespace URI or if Namespace processing is not being performed
`localName`	The local name (without prefix), or the empty string if Namespace processing is not being performed.
`qName`	The qualified name with prefix; empty string if not available.
`atts`	The attributes attached to the element; empty object if no attributes.

6

XML Class Generation for Java

The XML Class Generator for Java is contained within the oracle.xml.classgen package.

This chapter describes APIs in the following classes:

- CGDocument Class
- CGNode Class
- CGXSDElement Class
- DTDClassGenerator Class
- InvalidContentException Class
- oracg Class
- SchemaClassGenerator Class

See Also:

- *Oracle Application Developer's Guide - XML*

CGDocument Class

This class serves as the base document class for the DTD class Generator generated classes.

Syntax

```
public abstract class CGDocument extends oracle.xml.classgen.CGNode implements
java.io.Externalizable
```

Table 6–1 Summary of Methods of CGDocument

Method	Description
CGDocument() on page 6-2	Constructor for the root element of the DTD.
print() on page 6-2	Prints the constructed XML document.
readExternal() on page 6-3	Reads the compressed stream and creates the object corresponding to the root element.

CGDocument()

Constructor for the root element of the DTD.

Syntax

```
protected  CGDocument( String doctype,
                       oracle.xml.parser.v2.DTD dtd);
```

Parameter	Description
`doctype`	Name of the root Element of the DTD.
`dtd`	The DTD used to generate the classes.

print()

Prints the constructed XML Document. Throws `InvalidContentException` if the document's content does not match the grammar specified by DTD; the validation mode should be set to TRUE. See also `setValidationMode()` in DTDClassGenerator Class. The options are described in the following table.

Syntax	Description
`protected void print( java.io.OutputStream out);`	Prints the constructed XML Document to output stream.
`protected void print( java.io.OutputStream out, String enc);`	Prints the constructed XML Document to output stream with user-defined encoding.

Parameter	Description
`out`	Output stream to which the document will be printed.
`enc`	Encoding of the output stream.

readExternal()

Reads the compressed stream and creates the object corresponding to the root element. Used for instantiating the generated classes with XML instance document.

Syntax

```
protected void readExternal( java.io.ObjectInput inArg,
                          oracle.xml.comp.CXMLContext cxmlContext);
```

Parameter	Description
`in`	ObjectInput stream passed to read the compressed stream
`cxmlContext`	The context of the compressed stream

CGNode Class

This class serves as the base class for the classes corresponding to the nodes of XML document generated by the DTD class generator.

Syntax

```
public abstract class CGNode
```

Table 6–2 Fields of CGNode

Field	Syntax	Description
isValidating	protected boolean isValidating	Boolean to indicate the validating mode

Methods of CGNode

Table 6–3 Summary of Methods of CGNode

Method	Description
CGNode() on page 6-5	Constructor for the Elements of the DOM Tree.
addCDATASection() on page 6-5	Adds CDATA Section to the Element.
addData() on page 6-6	Adds PCDATA to the element node.
addNode() on page 6-6	Adds a node as a child to the element.
deleteData() on page 6-6	Deletes PCDATA from an element node.
getAttribute() on page 6-7	Retrieves the value of the attribute.
getCGDocument() on page 6-7	Retrieves the base document.
getData() on page 6-7	Retrieves the PCDATA of the element.
getDTDNode() on page 6-7	Retrieves the static DTD from the base document.
getElementNode() on page 6-8	Retrieves the XMLElement node corresponding to this CGNode.
getNode() on page 6-8	Retrieves the CGNode which is one of the children of the element corresponding to this node whose name matches the input string.

Table 6–3 Summary of Methods of CGNode (Cont.)

Method	Description
readExternal() on page 6-8	Reads the compressed stream and instantiates the corresponding node.
setAttribute() on page 6-8	Sets the value of the attribute.
setDocument() on page 6-9	Sets the base document.
setElementNode() on page 6-9	Sets the XMLElement node corresponding to this CGNode.
storeID() on page 6-9	Stores this value of ID identifier.
storeIDREF() on page 6-10	Stores this value for an IDREF identifier.
validateContent() on page 6-10	Checks if the content of the element is valid according to the Content Model specified in DTD.
validEntity() on page 6-10	Checks if the ENTITY identifier is valid.
validID() on page 6-11	Checks if the ID identifier is valid.
validNMTOKEN() on page 6-11	Checks if the NMTOKEN identifier is valid.
writeExternal() on page 6-11	Writes the compressed stream corresponding to this node.

CGNode()

Constructor for the Elements of the DOM Tree.

Syntax

```
protected  CGNode( String elementName);
```

Parameter	Description
`elementName`	Name of the element.

addCDATASection()

Adds CDATA Section to the Element. Throws `InvalidContentException` if `theData` has illegal characters; validation must be set to TRUE. See also `setValidationMode()` in DTDClassGenerator Class.

Syntax

```
protected void addCDATASection( String theData);
```

Parameter	Description
theData	Text to be added as CDATA Section to the element.

addData()

Adds PCDATA to the element node. Throws `InvalidContentException` if `theData` has illegal characters; validation must be set to TRUE. See also `setValidationMode()` in DTDClassGenerator Class.

Syntax

```
protected void addData( String theData);
```

Parameter	Description
theData	Text to be added a to the element.

addNode()

Adds a node as a child to the element. Throws `InvalidContentException` if `theData` has illegal characters; validation must be set to TRUE. See also `setValidationMode()` in DTDClassGenerator Class.

Syntax

```
protected void addNode( CGNode theNode);
```

Parameter	Description
theNode	The node to be added as child.

deleteData()

Deletes PCDATA from the element node. Throws `InvalidContentException` if `theData` has illegal characters; validation must be set to TRUE. See also `setValidationMode()` in DTDClassGenerator Class.

Syntax

```
protected void deleteData( String theData);
```

Parameter	Description
theNode	Text to be deleted from an element.

getAttribute()

Returns the value of the attribute.

Syntax

```
protected String getAttribute( String attName);
```

Parameter	Description
attName	Name of the attribute.

getCGDocument()

Gets the base document (root Element).

Syntax

```
protected CGDocument getCGDocument();
```

getData()

Gets the PCDATA of the Element. Throws `InvalidContentException` if the data is not present.

Syntax

```
protected String getData();
```

getDTDNode()

Retrieves the static DTD from the base CGDocument.

Syntax

```
protected abstract oracle.xml.parser.v2.DTD getDTDNode();
```

getElementNode()

Retrieves the XMLElement node corresponding to this CGNode.

Syntax

```
protected oracle.xml.parser.v2.XMLElement getElementNode();
```

getNode()

Retrieves the CGNode which is one of the children of the element corresponding to the node whose name matches the input string.

Syntax

```
protected java.lang.Object getNode(String theNode);
```

Parameter	Description
`theNode`	The name of the string corresponding to the CGNode returned.

readExternal()

Reads the compressed stream and instantiate the corresponding node. Throws the following exceptions:

- `IOException` when an I/O Error occurs
- `ClassNotFoundException` when the class could not be instantiated

Syntax

```
protected void readExternal( oracle.xml.io.XMLObjectInput in,
                             oracle.xml.comp.CXMLContext cxmlContext)
```

Parameter	Description
`in`	The `XMLObjectInput` stream that is used to read the compressed stream.
`cxmlContext`	The context of the compressed stream.

setAttribute()

Sets the value of the attribute.

Syntax

```
protected void setAttribute( String attName,
                             String value);
```

Parameter	Description
attName	Name of the attribute.
value	Value of the attribute.

setDocument()

Sets the base document (root element).

Syntax

```
public void setDocument( CGDocument d);
```

Parameter	Description
d	Base CGDocument.

setElementNode()

Sets the XMLElement node corresponding to this CGNode.

Syntax

```
protected void setElementNode( oracle.xml.parser.v2.XMLElement node);
```

Parameter	Description
node	The XMLElement.

storeID()

Store this value for an ID identifier, which can be verified with IDREF values.

Syntax

```
protected void storeID( String attName,
                        String id);
```

Parameter	Description
attName	Name of the ID attribute.
id	Value of the ID

storeIDREF()

Store this value for an IDREF identifier, which can be verified by the corresponding ID.

Syntax

```
protected void storeIDREF( String attName,
                           String idref);
```

Parameter	Description
attName	Name of the IDREF attribute.
idref	Value of the IDREF

validateContent()

Checks if the content of the element is valid according to the Content Model specified in DTD.

Syntax

```
protected void validateContent();
```

validEntity()

Checks if the ENTITY identifier is valid. Returns TRUE if ENTITY is valid, FALSE otherwise.

Syntax

```
protected boolean validEntity( String entity);
```

Parameter	Description
name	Value of the ENTITY attribute

validID()

Checks if the ID identifier is valid. Returns `TRUE` if ID is valid, `FALSE` otherwise.

Syntax

```
protected boolean validID( String name);
```

Parameter	Description
`name`	Value of the ID attribute.

validNMTOKEN()

Checks if the NMTOKEN identifier is valid. Returns `TRUE` if NMTOKEN is valid, `FALSE` otherwise.

Syntax

```
protected boolean validNMTOKEN( String name);
```

Parameter	Description
`name`	Value of the NMTOKEN attribute.

writeExternal()

Writes the compressed stream corresponding to this node.

Syntax

```
protected void writeExternal( oracle.xml.io.XMLObjectOutput out,
                 oracle.xml.comp.CXMLContext cxmlContext);
```

Parameter	Description
`out`	ObjectOutput stream to write the compressed data.
`cxmlContext`	The context of the compressed stream.

CGXSDElement Class

This class serves as the base class for the all the generated classes corresponding to the XML Schema generated by Schema Class Generator

Syntax

```
public abstract class CGXSDElement extends java.lang.Object
```

Table 6–4 Fields of CGXSDElement

Field	Syntax	Description
type	protected java.lang.Object type	Type information of a node

Table 6–5 Summary of Methods of CGXSDElement

Method	Description
CGXSDElement() on page 6-12	Default constructor.
addAttribute() on page 6-13	Adds the attribute of a given node to the hashtable.
addElement() on page 6-13	Adds the local elements of an element node to the vector corresponded to the elements.
getAttributes() on page 6-13	Returns the attributes as a hashtable of attribute names and values.
getChildElements() on page 6-13	Retrieves the vector of all local elements.
getNodeValue() on page 6-13	Returns the value of the node.
print() on page 6-14	Prints an element node.
printAttributes() on page 6-14	Prints an attribute node.
setNodeValue() on page 6-14	Sets the node value of an element.

CGXSDElement()

Default constructor.

Syntax

```
public  CGXSDElement();
```

addAttribute()

Adds the attribute of a given node to the hashtable.

Syntax

```
protected void addAttribute( String attName, java.lang.Object attValue);
```

Parameter	Description
attName	The attribute name.
attValue	The attribute value.

addElement()

Adds the local elements of an element node to the vector corresponded to the elements.

Syntax

```
protected void addElement( java.lang.Object elem);
```

Parameter	Description
elem	The object which needs to be added.

getAttributes()

Returns the attributes as a hashtable of attribute names and values.

Syntax

```
public java.util.Hashtable getAttributes();
```

getChildElements()

Retrieves the vector of all local elements.

Syntax

```
public java.util.Vector getChildElements();
```

getNodeValue()

Returns the value of the node.

Syntax

```
public String getNodeValue();
```

print()

Prints an element node. Throws an `IOException` if not able to print to the output stream

Syntax

```
public void print( oracle.xml.parser.v2.XMLOutputStream out);
```

Parameter	Description
`out`	The XMLObjectOutput stream to which the output is printed.

printAttributes()

Prints an attribute node. Throws an `IOException` if not able to print to the `XMLObjectOutput` stream.

Syntax

```
public void printAttributes( oracle.xml.parser.v2.XMLOutputStream out,
                             String name,  String namespace);
```

Parameter	Description
`out`	The XMLObjectOutput stream to which the output is printed.
`name`	The attribute name
`namespace`	The namespace

setNodeValue()

Sets the node value of an element.

Syntax

```
protected void setNodeValue( String value);
```

Parameter	Description
value	The node vale.

DTDClassGenerator Class

Generates the data binding classes corresponding to a DTD or an XML file based on a DTD.

Syntax

```
public class DTDClassGenerator extends java.lang.Object
```

Table 6–6 Summary of Methods of DTDClassGenerator

Method	Description
DTDClassGenerator() on page 6-16	Default constructor for DTDClassGenerator.
generate() on page 6-16	Traverses the DTD with element `doctype` as root and generates Java classes.
setGenerateComments() on page 6-17	Sets the switch to determine whether to generate java doc comments for the generated classes.
setJavaPackage() on page 6-17	Sets the package for the classes generated.
setOutputDirectory() on page 6-17	Sets the output directory where the java source code for the DTD is generated.
setSerializationMode() on page 6-18	Sets the switch to determine if the DTD should be saved as a serialized object or as text file.
setValidationMode() on page 6-18	Sets the switch to determine whether the classes generated should validate the XML document.

DTDClassGenerator()

Default constructor for DTDClassGenerator.

Syntax

```
public  DTDClassGenerator();
```

generate()

Traverses the DTD with element `doctype` as root and generates Java classes.

Syntax

```
public void generate( oracle.xml.parser.v2.DTD dtd,
                      String doctype);
```

Parameter	Description
DTD	The DTD used to generate the classes.
doctype	Name of the root element.

setGenerateComments()

Sets the switch to determine whether to generate java doc comments for the generated classes. Default value is TRUE.

Syntax

```
public void setGenerateComments( boolean comments);
```

Parameter	Description
comments	The boolean flag for turning on/off the java doc comment generation.

setJavaPackage()

Sets the package for the classes generated. Default - no package set.

Syntax

```
public void setJavaPackage( java.util.Vector packageName);
```

Parameter	Description
packageName	Name of the package.

setOutputDirectory()

Sets the output directory where the java source code for the DTD is generated. Default value is the current directory.

Syntax

```
public void setOutputDirectory( String dir);
```

Parameter	Description
dir	Output directory.

setSerializationMode()

Sets the switch to determine if the DTD should be saved as a serialized object or as text file. Serializing the DTD improves the performance when the generated classes are used to author XML files.

Syntax

```
public void setSerializationMode( boolean yes);
```

Parameter	Description
yes	The boolean flag for turning on/off saving of DTD as serialized object (TRUE). Default is saving as a text file (FALSE).

setValidationMode()

Sets the switch to determine whether the classes generated should validate the XML document being constructed. Default value is TRUE.

Syntax

```
public void setValidationMode( boolean yes);
```

Parameter	Description
yes	The boolean flag for turning on/off validation of XML document. Default is TRUE.

InvalidContentException Class

Defines the Exception thrown by DTD ClassGenerator and Schema Class Generator.

Syntax

```
public class InvalidContentException extends java.lang.Exception
```

InvalidContentException()

Constructor. The options are described in the following table.

Syntax	Description
`public InvalidContentException();`	Default constructor.
`public InvalidContentException( String s);`	This constructor takes an input String of information about the exception.

Parameter	Description
`s`	String that contains the information about the exception.

oracg Class

Provides a command-line interface to generate java classes corresponding to the DTD or XML

Syntax

```
public class oracg extends java.lang.Object
```

Table 6–7 Command-line options of oracg

Option	Description
`-help`	Prints the help message text.
`-version`	Prints the release version.
`-dtd [-root <rootName>]`	The input file is a DTD file or DTD based XML file.
`-schema <Schema File>`	The input file is a Schema file or Schema based XML file.
`-outputDir <Output Dir>`	The directory name where java source is generated.
`-package <Package Name>`	The package name(s) of the generated java classes.
`-comment`	Generate comments for the generated java source code.

SchemaClassGenerator Class

This class generates the classes corresponding to an XML Schema.

Syntax

```
public class SchemaClassGenerator extends java.lang.Object
```

Table 6–8 Summary of Methods of SchemaClassGenerator

Method	Description
SchemaClassGenerator() on page 6-21	Constructor.
generate() on page 6-22	Generates the Schema classes corresponding to top level elements, simpleType elements and complexType elements.
setGenerateComments() on page 6-22	Sets the switch to determine whether to generate java doc comments.
setJavaPackage() on page 6-22	Assigns a user-defined Java package name for each namespace.
setOutputDirectory() on page 6-23	Sets the output directory where the java source code for the Schema class are generated.

SchemaClassGenerator()

Constructor. The options are described in the following table.

Syntax	Description
`public SchemaClassGenerator();`	Default empty constructor for Schema Class Generator.
`public SchemaClassGenerator( String fileName)`	This constructor takes an input String containing the description of the XML Schema.

Parameter	Description
fileName	The input XML Schema.

generate()

Generates the Schema classes corresponding to top level elements, simpleType elements and complexType elements by calling createSchemaClass() on each of these nodes.

Syntax

```
public void generate( oracle.xml.parser.schema.XMLSchema schema);
```

Parameter	Description
XML	Schema object.

setGenerateComments()

Sets the switch to determine whether to generate java doc comments. TRUE by default.

Syntax

```
public void setGenerateComments( boolean comments)
```

Parameter	Description
comments	Turns on/off the java doc comment generation. TRUE by default.

setJavaPackage()

Assigns user-defined Java package name for each namespace. The Namespaces defined in the schema are queried, and their number should match the number of package names provided by the user; otherwise, an error is thrown.

Syntax

```
public void setJavaPackage( oracle.xml.parser.schema.XMLSchema schema,
                            java.util.Vector pkgName);
```

Parameter	Description
schema	The XML Schema
pkgName	A vector containing user defined package names given through command line.

setOutputDirectory()

Sets the output directory where the java source code for the Schema class are generated. The current directory is the default.

Syntax

```
public void setOutputDirectory( String dir);
```

Parameter	Description
dir	The output directory.

7

XML SQL Utility for Java

XML SQL Utility for Java (XSU) generates and stores XML from SQL queries.

This chapter contains descriptions of the following XSU classes:

- OracleXMLQuery Class
- OracleXMLSave Class
- OracleXMLSQLException Class
- OracleXMLSQLNoRowsException Class

See Also:

- *Oracle Application Developer's Guide - XML*

OracleXMLQuery Class

The OracleXMLQuery class generates XML given an SQL query.

Syntax

```
public class OracleXMLQuery extends java.lang.Object
```

Table 7–1 Summary of Fields of OracleXMLQuery

Field	Syntax	Description
DTD	public static final int DTD	Specifies that the DTD is to be generated
ERROR_TAG	public static final String ERROR_TAG	Specifies the default tag name for the ERROR document
MAXROWS_ALL	public static final int MAXROWS_ALL	Specifies that all rows be included in the result
NONE	public static final int NONE	Specifies that no DTD is to be generated
ROW_TAG	public static final String ROW_TAG	Specifies the default tag name for the ROW elements
ROWIDATTR_TAG	public static final String ROWIDATTR_TAG	Specifies the default tag name for the ROW elements
ROWSET_TAG	public static final String ROWSET_TAG	Specifies the default tag name for the document
SCHEMA	public static final int SCHEMA	Specifies that an XML schema is to be generated
SKIPROWS_ALL	public static final int SKIPROWS_ALL	Specifies that all rows be skipped in the result.

Table 7–2 Summary of Methods of OracleXMLQuery

Method	Description
OracleXMLQuery() on page 7-4	Class constructor.
close() on page 7-5	Closes open resources created by the Oracle XML engine.

Table 7–2 (Cont.) Summary of Methods of OracleXMLQuery

Method	Description
getNumRowsProcessed() on page 7-5	Returns the number of rows processed.
getXMLDOM() on page 7-5	Transforms data into an XML document.
getXMLMetaData() on page 7-6	Transforms object-relational data, specified in the constructor, into an XML document.
getXMLSAX() on page 7-7	Returns the DTD or XMLSchema for the XML document.
getXMLSchema() on page 7-7	Transforms object-relational data, specified in the constructor, into an XML document.
getXMLString() on page 7-7	Generates XMLSchema(s) corresponding to the specified query.
keepObjectOpen() on page 7-8	Transforms object-relational data, specified in the constructor, into an XML document.
removeXSLTParam() on page 7-8	Has the effect of turning on and off the persistency of objects from which XML data is retrieved.
setCollIdAttrName() on page 7-9	Removes the value of a top-level stylesheet parameter.
setDataHeader() on page 7-9	Sets the name of the id attribute of the collection element's separator tag.
setDateFormat() on page 7-9	Sets the XML data header.
setEncoding() on page 7-10	Sets the format of the generated dates in the XML doc.
setErrorTag() on page 7-10	Sets the encoding processing instruction in the XML doc.
setException() on page 7-10	Sets the tag to be used to enclose the XML error docs.
setMaxRows() on page 7-11	Allows the user to pass in an exception to be handled by the XSU.
setMetaHeader() on page 7-11	Sets the maximum number of rows to be converted to XML.
setRaiseException() on page 7-11	Sets the XML meta header.
setRaiseNoRowsException() on page 7-12	Instructs the XSU whether to throw the raised exceptions.
setRowIdAttrName() on page 7-12	Instructs the XSU whether to throw an `OracleXMLNoRowsException` when the generated XML doc is empty.

Table 7–2 (Cont.) Summary of Methods of OracleXMLQuery

Method	Description
setRowIdAttrValue() on page 7-12	Sets the name of the id attribute of the row enclosing tag.
setRowsetTag() on page 7-13	Specifies the scalar column whose value will be assigned to the id attribute of the row enclosing tag.
setRowTag() on page 7-13	Sets the tag to be used to enclose the XML dataset.
setSkipRows() on page 7-13	Sets the number of rows to skip.
setSQLToXMLNameEscaping() on page 7-13	Has the effect of turning on and off the escaping of XML tags in cases where mapped SQL object name would not make a valid XML identifier.
setStylesheetHeader() on page 7-14	Sets the stylesheet header.
setXSLT() on page 7-14	Registers an XSL transform to be applied to the generated XML.
setXSLTParam() on page 7-15	Sets the value of a top-level stylesheet parameter.
useLowerCaseTagNames() on page 7-15	Sets the tag names to lower case.
useNullAttributeIndicator() on page 7-15	Specifies if `NULL`ness is indicated by a special XML attribute or by omitting the entity from the XML document.
useTypeForCollElemTag() on page 7-16	Instructs the XSU to use the collection element's type name as the collection element's tag name.
useUpperCaseTagNames() on page 7-16	Sets the tag names to upper case.

OracleXMLQuery()

Class constructor for the OracleXMLQueryObject. The options are described in the following table.

Syntax	Description
`public OracleXMLQuery( java.sql.Connection conn, java.sql.ResultSet rset);`	Creates an OracleXMLQuery from a databse connection and a jdbc result set object.

Syntax	Description
`public OracleXMLQuery( java.sql.Connection conn, String query);`	Creates an OracleXMLQuery from a databse connection and an SQL query string.
`public OracleXMLQuery( oracle.xml.sql.dataset.OracleXMLDataSet dset);`	Creates an OracleXMLQuery from a dataset.

Parameter	Description
`conn`	database connection
`rset`	jdbc result set object
`query`	the SQL query string
`dset`	dataset

close()

Closes any open resource, created by the OracleXML engine. This will not close for instance result set supplied by the user.

Syntax

```
public void close();
```

getNumRowsProcessed()

Returns the number of rows processed.

Syntax

```
public long getNumRowsProcessed();
```

getXMLDOM()

Transforms the object-relational data, specified in the constructor, into XML. Returns a representation of the XML document. The options are described in the following table.

Syntax	Description
`public org.w3c.dom.Document getXMLDOM();`	Returns a DOM representation of the XML document.

Syntax	Description
`public org.w3c.dom.Document getXMLDOM( int metaType);`	The argument is used to specify the type of XML metadata the XSU is to generate along with the XML. Currently this value is ignored, and no XML metadata is generated. Returns a string representation of the XML document.
`public org.w3c.dom.Document getXMLDOM( org.w3c.dom.Node root);`	If not `NULL`, the argument is considered the "root" element of the XML doc. Returns the string representation of the XML document.
`public org.w3c.dom.Document getXMLDOM( org.w3c.dom.Node root, int metaType);`	If not `NULL`, the `root` argument is considered the "root" element of the XML doc. The `metaType` argument is used to specify the type of XML metadata the XSU is to generate along with the XML. Currently this value is ignored, and no XML metadata is generated. Returns the string representation of the XML document.

Parameter	Description
`metaType`	the type of XML metadata (NONE, SCHEMA)
`root`	root node to which to append the new XML

getXMLMetaData()

This functions returns the DTD or the XMLSchema for the XML document which would have been generated by a `getXML*()` call, such as `getXMLDOM()`, `getXMLSAX()`, `getXMLSchema()`, or `getXMLString()`.

Syntax

```
public String getXMLMetaData( int metaType,
                              boolean withVer);
```

Parameter	Description
`metaType`	Specifies the type of XML metadata to be generated (NONE or DTD).
`withVer`	Specifies whether to generate the version processing instruction

getXMLSAX()

Transforms the object-relational data, specified in the constructor, into an XML document.

Syntax

```
public void getXMLSAX( org.xml.sax.ContentHandler sax);
```

Parameter	Description
sax	ContentHandler object to be registered.

getXMLSchema()

This methods generates the XML Schema(s) corresponding to the specified query; returns the XML Schema(s).

Syntax

```
public org.w3c.dom.Document[] getXMLSchema();
```

getXMLString()

Transforms the object-relational data, specified in the constructor, into a XML document. Returns the string representation of the XML document. The options are described in the following table.

Syntax	Description
`public String getXMLString();`	Takes no arguments.
`public String getXMLString( int metaType);`	The `metaType` argument is used to specify the type of XML metadata the XSU is to generate along with the XML.
`public String getXMLString( org.w3c.dom.Node root);`	If not `NULL`, the `root` argument, is considered the root element of the XML doc.
`public String getXMLString( org.w3c.dom.Node root, int metaType);`	If not `NULL`, the `root` argument is considered the root element of the XML doc. The `metaType` argument is used to specify the type of XML metadata the XSU is to generate along with the XML. Note that if the `root` argument is non-`NULL`, no DTD is generated even if requested.

Parameter	Description
metaType	The type of XML metadata (NONE, DTD, or SCHEMA, static fields of this class)
root	root node to which to append the new XML

keepObjectOpen()

The default behavior for all the getXML*() functions which DO NOT TAKE in a ResultSet object, such as getXMLDOM(), getXMLSAX(), getXMLSchema(), or getXMLString(), is to close the ResultSet object and Statement objects at the end of the call. If the persistent feature is needed, where by calling getXML() repeatedly the next set of rows is obtained, this behavior must be turned off by calling this function with value TRUE. OracleXMLQuery would not close the ResultSet and Statement objects after the getXML() calls. To close the cursor state, the close() function must be called explicitly.

Syntax

```
public void keepObjectOpen( boolean alive);
```

Parameter	Description
alive	Should the object be kept open?

removeXSLTParam()

Removes the value of a top-level stylesheet parameter. NOTE: if no stylesheet is registered, this method is a no op.

Syntax

```
public void removeXSLTParam( String name);
```

Parameter	Description
name	Parameter name

setCollIdAttrName()

Sets the name of the `id` attribute of the collection element's separator tag. Passing `NULL` or an empty string causes the `row id` attribute to be omitted.

Syntax

```
public void setCollIdAttrName( String attrName);
```

Parameter	Description
`attrName`	Attribute Name

setDataHeader()

Sets the XML data header, the XML entity which is appended at the beginning of the query-generated XML entity, the rowset. The two entities are enclosed by the tag specified through the `docTag` argument. The last data header specified is the one that is used. Passing in `NULL` for the `header` parameter unsets the data header.

Syntax

```
public void setDataHeader( java.io.Reader header,
                           String docTag);
```

Parameter	Description
`header`	Header
`docTag`	Tag used to enclose the data header and the rowset

setDateFormat()

Sets the format of the generated dates in the XML doc. The syntax of the date format pattern, the date mask, should conform to the requirements of the java.text.SimpleDateFormat class. Setting the mask to `NULL` or an empty string, unsets the date mask.

Syntax

```
public void setDateFormat( String mask);
```

Parameter	Description
mask	The data mask

setEncoding()

Sets the encoding processing instruction (PI) in the XML doc. If NULL or an empty string are specified as the encoding, then the default characterset is specified in the encoding PI.

Syntax

```
public void setEncoding( String enc)
```

Parameter	Description
enc	Encoding of the CML doc (IANA name of encoding)

setErrorTag()

Sets the tag to be used to enclose the XML error docs.

Syntax

```
public void setErrorTag( String tag);
```

Parameter	Description
tag	Tag name

setException()

Allows the user to pass in an exception, and have the XSU handle it.

Syntax

```
public void setException( java.lang.Exception e);
```

Parameter	Description
e	The exception to be processed by XSU

setMaxRows()

Sets the maximum number of rows to be converted to XML. By default there is no maximum set. To explicitly specify no max, see `MAXROWS_ALL` field.

Syntax

```
public void setMaxRows( int rows);
```

Parameter	Description
`rows`	Maximum number of rows to generate

setMetaHeader()

Sets the XML meta header. When set, the header is inserted at the beginning of the metadata part (DTD or XMLSchema) of each XML document generated by this object. The last meta header specified is the one that is used. Setting the `header` to `NULL` or an empty string unsets the meta header.

Syntax

```
public void setMetaHeader( java.io.Reader header);
```

Parameter	Description
`header`	Header

setRaiseException()

Instructs the XSU whether to throw the raised exceptions. If this call isn't made, or if `FALSE` is passed to the `flag` argument, the XSU catches the SQL exceptions and generates an XML doc from the exception message.

Syntax

```
public void setRaiseException( boolean flag);
```

Parameter	Description
`flag`	Should the raised exception be thrown?

setRaiseNoRowsException()

Instructs the XSU whether to throw an `OracleXMLNoRowsException` when the generated XML doc is empty. By default, the exception is not thrown.

Syntax

```
public void setRaiseNoRowsException( boolean flag);
```

Parameter	Description
`flag`	Should the `OracleXMLNoRowsException` be thrown if no data found?

setRowIdAttrName()

Sets the name of the id attribute of the row enclosing tag. Passing `NULL` or an empty string causes the row id attribute to be omitted.

Syntax

```
public void setRowIdAttrName( String attrName);
```

Parameter	Description
`attrName`	Attribute name

setRowIdAttrValue()

Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag. Passing `NULL` or an empty string causes the row id attribute to be assigned the row count value, such as 0, 1, 2, and so on.

Syntax

```
public void setRowIdAttrValue( String colName);
```

Parameter	Description
`colName`	Column whose value will be assigned to the row id attribute

setRowsetTag()

Sets the tag to be used to enclose the XML dataset.

Syntax

```
public void setRowsetTag( String tag);
```

Parameter	Description
tag	Tag name

setRowTag()

Sets the tag to be used to enclose the XML element corresponding to a db. record.

Syntax

```
public void setRowTag( String tag);
```

Parameter	Description
tag	Tag name.

setSkipRows()

Sets the number of rows to skip. By default 0 rows are skipped. To skip all the rows use `SKIPROWS_ALL`.

Syntax

```
public void setSkipRows( int rows);
```

Parameter	Description
rows	Number of rows to skip.

setSQLToXMLNameEscaping()

This turns on or off escaping of XML tags in the case that the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.

Syntax

```
public void setSQLToXMLNameEscaping( boolean flag);
```

Parameter	Description
flag	Whether to turn on SQL to XML identifier escaping.

setStylesheetHeader()

Sets the stylesheet header, which contains stylesheet processing instructions, in the generated XML doc. Passing NULL in the argument will unset the stylesheet header and the stylesheet type. The options are described in the following table.

Syntax	Description
`public void setStylesheetHeader( String uri);`	Sets stylesheet header using the stylesheet URI.
`public void setStylesheetHeader( String uri, String type);`	Sets the stylesheet header using the stylesheet URI and the stylesheet type.

Parameter	Description
uri	Stylesheet URI
type	Stylesheet type; defaults to text/xsl

setXSLT()

Registers a XSL transform to be applied to generated XML. If a stylesheet is already registered, it is replaced by the new one. To un-register the stylesheet, pass in NULL value for the argument. The options are described in the following table.

Syntax	Description
`public void setXSLT( java.io.Reader stylesheet, String ref);`	The stylesheet parameter is passed in as the data.

Syntax	Description
`public void setXSLT( java.lang.String stylesheet, String ref);`	The stylesheet parameter is passed in as a URI to the document.

Parameter	Description
`stylesheet`	The stylesheet.
`ref`	URL for include, import and external entities.

setXSLTParam()

Sets the value of a top-level stylesheet parameter. The parameter value is expected to be a valid XPath expression; therefore the string literal values have to be explicitly quoted). If no stylesheet is registered, this method is a no op.

Syntax

```
public void setXSLTParam( String name,
                          String value);
```

Parameter	Description
`name`	Parameter name
`value`	Parameter value as an XPATH expression

useLowerCaseTagNames()

This will set the case to be lower for all tag names. Note, make this call after all the desired tags have been set.

Syntax

```
public void useLowerCaseTagNames();
```

useNullAttributeIndicator()

Specifies if `NULL`ness is indicated by a special XML attribute or by omitting the entity from the XML document.

Syntax

```
public void useNullAttributeIndicator( boolean flag);
```

Parameter	Description
flag	Should the attribute be used to indicate NULL?

useTypeForCollElemTag()

By default, the tag name for elements of a collection is the collection's tag name followed by "_item". This method, when called with argument value of TRUE, instructs the XSU to use the collection element's type name as the collection element's tag name.

Syntax

```
public void useTypeForCollElemTag( boolean flag);
```

Parameter	Description
flag	Should the column element type be used to indicate its tag name?

useUpperCaseTagNames()

Sets all tag names to upper case. This call should be made only after all the desired tags have been set.

Syntax

```
public void useUpperCaseTagNames();
```

OracleXMLSave Class

OracleXMLSave class supports canonical mapping from XML to object-relational tables or views. It supports inserts, updates and deletes. The user first creates the class by passing in the table name on which these DML operations need to be done. After that, the user is free to use the insert/update/delete on this table.

Many useful functions are provided in this class to help in identifying the key columns for update or delete and to restrict the columns being updated.

Syntax

```
public class OracleXMLSave extends java.lang.Object
```

Table 7–3 Summary of Fields of OracleXMLSave

Field	Syntax	Description
DATE_FORMAT	public static final String DATE_FORMAT	The date format for use in setDateFormat.
DEFAULT_BATCH_SIZE	public static int DEFAULT_BATCH_SIZE	Default insert batch size is 17.
xDocIsEsc	public boolean xDocIsEsc	Indicates whether or not the xml doc has undergone SQL to XML escaping.

Table 7–4 Summary of Methods of OracleXMLSave

Method	Description
OracleXMLSave() on page 7-18	The public constructor for the Save class.
close() on page 7-19	It closes/deallocates all the context associated with this object.
deleteXML() on page 7-19	Deletes the rows in the table based on the XML document.
getURL() on page 7-20	Return a URL object given a file name or a URL.
insertXML() on page 7-20	Inserts an XML document into the specified table.
removeXSLTParam() on page 7-21	Removes the value of a top-level stylesheet parameter.

Table 7–4 (Cont.) Summary of Methods of OracleXMLSave

Method	Description
setBatchSize() on page 7-22	Changes the batch size used during DML operations.
setCommitBatch() on page 7-22	Sets the commit batch size.
setDateFormat() on page 7-22	Describes to the XSU the format of the dates in the XML document.
setIgnoreCase() on page 7-23	Instructs the XSU to perform a case-insensitive match of XML elements to database columns or attributes.
setKeyColumnList() on page 7-23	Sets the list of columns to be used for identifying a particular row in the database table during update or delete.
setPreserveWhitespace() on page 7-24	Instructs the XSU whether to preserve whitespaces.
setRowTag() on page 7-24	Names the tag used in the XML doc. to enclose the XML elements corresponding to each row value.
setSQLToXMLNameEscaping() on page 7-24	This turns on or off escaping of XML tags when an SQL object name would not make a valid XML identifier.
setUpdateColumnList() on page 7-25	Set the column values to be updated.
setXSLT() on page 7-25	Registers a XSL transform to be applied to generated XML.
setXSLTParam() on page 7-26	Sets the value of a top-level stylesheet parameter.
updateXML() on page 7-26	Updates the table given the XML document.

OracleXMLSave()

The public constructor for the OracleXMLSave class.

Syntax

```
public  OracleXMLSave( java.sql.Connection oconn,
                       String tabName;
```

Parameter	Description
`oconn`	Connection object (connection to the database)
`tableName`	The name of the table that should be updated

close()

Closes/deallocates all the context associated with this object.

Syntax

```
public void close();
```

deleteXML()

Deletes the rows in the table based on the XML document. Returns the number of XML ROW elements processed. This may or may not be equal to the number of database rows deleted based on whether the rows selected through the XML document uniquely identified the rows in the table.

By default, the delete processing matches all the element values with the corresponding column names. Each ROW element in the input document is taken as a separate delete statement on the table. By using the `setKeyColumnList()`, the list of columns that must be matched to identify the row to be deleted is set, and other elements are ignored. This is an efficient method for deleting more than one row in the table if matching is employed (since the delete statement is cached). Otherwise, a new delete statement has to be created for each ROW element in the input document. The options are described in the following table.

Syntax	Description
`public int deleteXML( org.w3c.dom.Document doc);`	XML document is in DOM form.
`public int deleteXML( java.io.InputStream xmlStream);`	XML document is in Stream form.
`public int deleteXML( java.io.Reader xmlReader);`	XML document is in Reader form.
`public int deleteXML( String xmlDoc);`	XML document is in String form.
`public int deleteXML ( java.net.URL url);`	XML document is accessed through the URL.

Parameter	Description
`doc`	The XML document in DOM form.

Parameter	Description
xmlStream	The XML document in Stream form.
xmlReader	The XML document in Reader form.
xmlDoc	The XML document in String form.
url	The URL to the document to use to delete the rows in the table.

getURL()

Returns a URL object identifying the target entity, given a file name or a URL. If the argument passed is not in a valid URL format, such as "http://..." or "file://...", then this method attempts to correct the argument by pre-pending "file://". If a NULL or an empty string are passed to it, NULL is returned.

Syntax

```
public static java.net.URL getURL( String target);
```

Parameter	Description
target	File name or URL string.

insertXML()

Inserts an XML document into the specified table. Returns the number of rows inserted.

- Inserts the values into the table by matching the element name with the column name, and inserts a NULL value for all elements that are missing in the input document. By using the setUpdateColumnList(), no NULL values would be inserted for the rest of the columns; instead, default values would be used.
- To set the list of all key column, use setKeyColumnList().
- To set the list of columns to update, use setUpdateColumnList().

The options are described in the following table.

Syntax	Description
`public int insertXML( org.w3c.dom.Document doc);`	Inserts an XML document from a DOM.
`public int insertXML( java.io.InputStream xmlStream);`	Inserts an XML document from an InputStream.
`public int insertXML( java.io.Reader xmlStream);`	Inserts an XML document from a Reader.
`public int insertXML( String xmlDoc);`	Inserts an XML document from a String.
`public int insertXML( java.net.URL url);`	Inserts an XML document from a URL.

Parameter	Description
`doc`	DOM for inserting rows into the table.
`xmlStream`	Stream of data used for inserting rows into the table.
`xmlDOC`	String used for inserting rows into the table.
`url`	The URL to the document used for inserting rows into the table.

removeXSLTParam()

Removes the value of a top-level stylesheet parameter. If no stylesheet is registered, this method is a no op.

Syntax

```
public void removeXSLTParam( String name);
```

Parameter	Description
`name`	Parameter name

setBatchSize()

Changes the batch size used during DML operations. When performing inserts, updates or deletes, it is recommended to batch the operations to minimize I/O cycles; however, this requires more cache for storing the bind values while the operations are executing. When batching is used, the commits occur only in terms of batches. If a single statement inside a batch fails, the entire batch is rolled back. If this behavior is undesirable, set batch size to 1. The default batch size is `DEFAULT_BATCH_SIZE`.

Syntax

```
public void setBatchSize( int size);
```

Parameter	Description
`size`	The batch size to use for all DML.

setCommitBatch()

Sets the commit batch size, which refers to the number of records inserted after which a commit must follow. If `size < 1`, or the session is in "auto-commit" mode, the XSU does not make any explicit commits. Default commit-batch size is `0`.

Syntax

```
public void setCommitBatch( int size);
```

Parameter	Description
`size`	Commit batch size.

setDateFormat()

Describes to the XSU the format of the dates in the XML document. By default, OracleXMLSave assumes that the date is in format 'MM/dd/yyyy HH:mm:ss'. You can override this default format by calling this function. The syntax of the date format pattern, the date mask, should conform to the requirements of the java.text.SimpleDateFormat class. Setting the mask to `NULL` or an empty string, causes the use of the default mask -- `OracleXMLSave.DATE_FORMAT`.

Syntax

```
public void setDateFormat( String mask);
```

Parameter	Description
mask	The date mask.

setIgnoreCase()

The XSU performs mapping of XML elements to database columns or attributes based on the element names (XML tags). This function instructs the XSU to perform a case-insensitive match. This may affect the metadata caching performed when creating the Save object.

Syntax

```
public void setIgnoreCase( boolean ignore);
```

Parameter	Description
flag	Should the tag case in the XML doc be ignored?

setKeyColumnList()

Sets the list of columns to be used for identifying a particular row in the database table during update or delete. This call is ignored for the insert case. The key columns must be set before updates can be done. It is optional for deletes. When this key columns is set, then the values from these tags in the XML document is used to identify the database row for update or delete. Currently, there is no way to update the values of the key columns themselves, since there is no way in the XML document to specify that case.

Syntax

```
public void setKeyColumnList( String[] keyColNames);
```

Parameter	Description
keyColNames	The names of the list of columns that are used as keys.

setPreserveWhitespace()

Instructs the XSU whether to preserve whitespaces.

Syntax

```
public void setPreserveWhitespace( boolean flag);
```

Parameter	Description
flag	Should the whitespaces be preserved?

setRowTag()

Names the tag used in the XML doc so to enclose the XML elements corresponding to each row value. Setting the value of this to NULL implies that there is no row tag present, and the top level elements of the document correspond to the rows themselves.

Syntax

```
public void setRowTag( String rowTag);
```

Parameter	Description
tag	Tag name.

setSQLToXMLNameEscaping()

This turns on or off escaping of XML tags when the SQL object name, which is mapped to a XML identifier, is not a valid XML identifier.

Syntax

```
public void setSQLToXMLNameEscaping( boolean flag);
```

Parameter	Description
flag	Should the SQL to XML escaping be turned on?

setUpdateColumnList()

Set the column values to be updated. Applies to inserts and updates, not deletes.

- In case of insert, the default is to insert values to all the columns in the table.
- In case of updates, the default is to only update the columns corresponding to the tags present in the ROW element of the XML document. When specified, these columns alone will get updated in the update or insert statement. All other elements in the document will be ignored.

Syntax

```
public void setUpdateColumnList( String[] updColNames);
```

Parameter	Description
updColNmaes	The string list of columns to be updated.

setXSLT()

Registers a XSL transform to be applied to generated XML. If a stylesheet was already registered, it gets replaced by the new one. To un-register the stylesheet pass in a NULL for the stylesheet argument. The options are described in the following table.

Syntax	Description
public void setXSLT(java.io.Reader stylesheet, String ref);	The stylesheet parameter is passed in as the data.
public void setXSLT(String stylesheet, String ref);	The stylesheet parameter is passed in as a URI to the document.

Parameter	Description
stylesheet	The stylesheet URI.
ref	URL for include, import and external entities.

setXSLTParam()

Sets the value of a top-level stylesheet parameter. The parameter value is expected to be a valid XPath expression (note that string literal values would therefore have to be explicitly quoted). If no stylesheet is registered, this method is a no op.

Syntax

```
public void setXSLTParam( String name,
                          String value);
```

Parameter	Description
`name`	Parameter name.
`value`	Parameter value as an XPATH expression.

updateXML()

Updates the table given the XML document. Returns the number of XML elements processed. This may or may not be equal to the number of database rows modified, depending on whether the rows selected through the XML document uniquely identify the rows in the table.

- The update requires a list of key columns used to uniquely identify rows to update. By default, the update uses the list of key columns and matches the values of the corresponding elements in the XML document to identify a particular row, updating all columns that have an equivalent element in the XML document. Each ROW element is treated as a separate update to the table.
- A a list of columns to update can be supplied to update only desired columns and ignore any other elements present in the XML document. This is a very efficient method, because if there are more than one row present in the input XML document, the update statement itself is cached and batched.
- To set the list of all key column, use `setKeyColumnList()`.
- To set the list of columns to update, use `setUpdateColumnList()`.

The options are described in the following table.

Syntax	Description
`public int updateXML( org.w3c.dom.Document doc);`	Updates the table given the XML document in a DOM tree form.
`public int updateXML( java.io.InputStream xmlStream);`	Updates the table given the XML document in a stream form.
`public int updateXML( java.io.Reader xmlStream);`	Updates the table given the XML document in a stream form.
`public int updateXML( String xmlDoc);`	Updates the table given the XML document in a string form.
`public int updateXML( java.net.URL url);`	Updates the columns in a database table, based on the element values in the supplied XML document.

Parameter	Description
`doc`	The DOM tree form of the XML document
`xmlStream`	The stream form of the XML document
`xmlDoc`	The string form of the XML document
`url`	The URL to the document to use to update the table

OracleXMLSQLException Class

Class for managing all exceptions thrown by the XSU.

Syntax

```
public class OracleXMLSQLException extends java.lang.RuntimeException
```

Table 7–5 Summary of Methods of OracleXMLSQLException

Method	Description
OracleXMLSQLException() on page 7-28	Creates a new OracleXMLSQLException.
getErrorCode() on page 7-29	Returns the SQL error code thrown.
getParentException() on page 7-29	Returns the original exception if there was one; otherwise, returns `NULL`.
getXMLErrorString() on page 7-30	Prints XML error string with given error tag name.
getXMLSQLErrorString() on page 7-30	Prints the SQL parameters in the error message.
setErrorTag() on page 7-30	Sets error tag used to generate XML error reports.

OracleXMLSQLException()

Creates a new OracleXMLSQLException. The options are described in the following table.

Syntax	Description
`public OracleXMLSQLException( Exception e);`	Sets the parent exception as passed in.
`public OracleXMLSQLException( Exception e, String errorTagName);`	Sets the error tag name as passed in.
`public OracleXMLSQLException( String message);`	Sets the error message to be returned.

Syntax	Description
`public OracleXMLSQLException( String message, j Exception e);`	Sets the parent exception and the error message to be returned.
`public OracleXMLSQLException( String message, Exception e, String errorTagName);`	Sets the error message, parent exception, and error tag to be used.
`public OracleXMLSQLException( String message, int errorCode);`	Sets the error message and SQL error code.
`public OracleXMLSQLException( String message, int errorCode, String errorTagName);`	Sets the eror message, SQL error code, and the error tag to be used.
`public OracleXMLSQLException( String message, String errorTagName);`	Sets the error message and the error tag to be used.

Parameter	Description
`e`	The exception.
`errorTagName`	The error tag name.
`message`	The error message.
`errorCode`	the SQL error code.

getErrorCode()

Returns the SQL error code thrown.

Syntax

```
public int getErrorCode();
```

getParentException()

Returns the original exception, if there was one; otherwise, returns `NULL`.

Syntax

```
public java.lang.Exception getParentException();
```

getXMLErrorString()

Prints the XML error string with the given error tag name.

Syntax

```
public String getXMLErrorString();
```

getXMLSQLErrorString()

Prints the SQL parameters as well in the error message.

Syntax

```
public String getXMLSQLErrorString();
```

setErrorTag()

Sets the error tag name, which is then used by `getXMLErrorString()` and `getXMLSQLErrorString() to` generate XML error reports.

Syntax

```
public void setErrorTag( String tagName);
```

Parameter	Description
`tagName`	The tag name of the error

OracleXMLSQLNoRowsException Class

The exception that can be thrown when no rows are found.

Syntax

```
public class OracleXMLSQLNoRowsException extends OracleXMLSQLException
```

OracleXMLSQLNoRowsException()

Creates a new `OracleXMLSQLNoRowsException`. The options are described in the following table.

Syntax	Description
`public OracleXMLSQLNoRowsException();`	Default class constructor.
`public OracleXMLSQLNoRowsException( String errorTag);`	Sets the error tag as the passed in argument.

Parameter	Description
`errorTag`	The error tag.

8

XSQL Pages Publishing Framework for Java

The XSQL Pages Publishing Framework is contained in the oracle.xml.xsql package, also known as Oracle XSQL Servlet.

This chapter describes the following sections:

- XSQLActionHandler Interface
- XSQLActionHandlerImpl Class
- XSQLPageRequest Interface
- XSQLParserHelper Class
- XSQLRequest Class
- XSQLRequestObjectListener Interface
- XSQLServletPageRequest Class
- XSQLStylesheetProcessor Class
- XSQLConnectionManager Interface
- XSQLConnectionManagerFactory Interface
- XSQLDocumentSerializer Interface

See Also:

- *Oracle Application Developer's Guide - XML*

oracle.xml.xsql Package

The Oracle XSQL Pages Publishing Framework engine. Table 8–1 summarizes the classes and interfaces of this package.

Table 8–1 Summary of Classes and Interfaces of oracle.xml.xsql

Class/Interface	Description
XSQLActionHandler Interface	Interface that must be implemented by all XSQL Action Element Handlers
XSQLActionHandlerImpl Class	Base Implementation of XSQLActionHandler that can be extended to create your own custom handlers.
XSQLPageRequest Interface	Interface representing a request for an XSQL Page
XSQLParserHelper Class	Common XML Parsing Routines
XSQLRequest Class	Programmatically process a request for an XSQL Page.
XSQLRequestObjectListener Interface	Interface that an object created by an action handler. Can implement to be notified when the current page request processing is completed.
XSQLServletPageRequest Class	Implementation of XSQLPageRequest for Servlet-based XSQL Page requests.
XSQLStylesheetProcessor Class	XSLT Stylesheet Processing Engine
XSQLConnectionManager Interface	Must be implemented to override the built-in connection manager implementation.
XSQLConnectionManagerFactory Interface	Must be implemented to override the built-in connection manager implementation.
XSQLDocumentSerializer Interface	Must be implemented by all XSQL Serializers which serialize an XSQL data page as an XML Document to a PrintWriter.

XSQLActionHandler Interface

Implemented by all XSQL Action Element Handlers. Upon encountering an XSQL Action Element of the form <xsql:xxxx> in an XSQL page, the XSQL Page Processor invokes the associated XSQL Action Handler by constructing an instance of the handler using the no-args constructor, and invoking the XSQL Action Handler's handleAction() method.

Syntax

```
public interface XSQLActionHandler
```

Table 8–2 Summary of Methods of XSDLAction Handler

Method	Description
handleAction() on page 8-3	Handle the action.
init() on page 8-3	Initializes the Action Handler.

handleAction()

Handle the action by executing code and appending new child DOM nodes to the root. The XSQL Page Processor replaces the action element in the XSQL Page being processed with the document fragment of nodes.

Syntax

```
public void handleAction( oracle.xml.xsql.Node rootNode);
```

Parameter	Description
rootNode	Root node of generated document fragment.

init()

Initializes the Action Handler.

Syntax

```
public void init( XSQLPageRequest env, oracle.xml.xsql.Element e);
```

Parameter	Description
env	XSQLPageRequest object.
e	DOM element representing the Action Element being handled.

XSQLActionHandlerImpl Class

Base Implementation of XSQLActionHandler that creates custom handlers. Includes a set of useful helper methods. If there is an intent to extend this class and override the `init()` method, a call must be made to `super.init(env,e)`.

Syntax

```
public abstract class XSQLActionHandlerImpl extends java.lang.Object implements
XSQLActionHandler Interface
```

Table 8–3 Summary of Methods of XSQLActionHandlerImpl

Method	Description
XSQLActionHandlerImpl() on page 8-5	Class constructor.
init() on page 8-5	Initializes the action handler.

XSQLActionHandlerImpl()

Class constructor.

Syntax

```
public  XSQLActionHandlerImpl();
```

init()

Initializes the action handler.

Syntax

```
public void init( XSQLPageRequest env, oracle.xml.xsql.Element e);
```

Parameter	Description
`env`	The XSQLPageRequest context.
`e`	The action element

XSQLPageRequest Interface

Interface representing a request for an XSQL Page.

Syntax

```
public interface XSQLPageRequest
```

Table 8–4 Summary of Methods of XSQLPageRequest

Method	Description
createNestedRequest() on page 8-8	Returns an instance of a nested Request
getConnectionName() on page 8-8	Returns the name of the connection being used for this request May be null if no connection set/in-use.
getErrorWriter() on page 8-8	Returns a PrintWriter to print out errors processing this request
getJDBCConnection() on page 8-9	Gets the JDBC connection being used for this request (can be null)
getPageEncoding() on page 8-9	Returns encoding of source XSQL Page associated with this request
getParameter() on page 8-9	Returns the value of the requested parameter
getPostedDocument() on page 8-9	Returns the content of Posted XML for this request as an XML Document
getRequestParamsAsXMLDocument() on page 8-9	Returns the content of a Request parameters as an XML Document
getRequestType() on page 8-9	Returns a string identifying the type of page request being made.
getSourceDocumentURI() on page 8-10	Returns a String representation of the requested document's URI
getStylesheetParameter() on page 8-10	Gets a stylesheet parameter by name
getStylesheetParameters() on page 8-10	Gets an enumeration of stylesheet parameter names
getStylesheetURI() on page 8-10	Returns the URI of the stylesheet to be used to process the result.

Table 8–4 (Cont.) Summary of Methods of XSQLPageRequest

Method	Description
getUserAgent() on page 8-10	Returns a String identifier of the requesting program
getWriter() on page 8-10	Returns a PrintWriter used for writing out the results of a page request
getXSQLConnection() on page 8-11	Gets the XSQLConnection Object being used for this request Might be null.
isIncludedRequest() on page 8-11	Returns true if this request is being included in another.
isOracleDriver() on page 8-11	Returns true if the current connection uses the Oracle JDBC Driver
printedErrorHeader() on page 8-11	Returns the state of whether an Error Header has been printed
requestProcessed() on page 8-11	Allows Page Request to Perform end-of-request processing
setConnectionName() on page 8-11	Sets the connection name to use for this request
setContentType() on page 8-12	Sets the content type of the resulting page
setIncludingRequest() on page 8-12	Sets the Including Page Request object for this request.
setPageEncoding() on page 8-12	Sets encoding of source XSQL page associated with this request.
setPageParam() on page 8-13	Sets a dynamic page parameter value.
setPostedDocument() on page 8-13	Allows programmatic setting of the Posted Document
setPrintedErrorHeader() on page 8-13	Sets whether an Error Header has been printed
setStylesheetParameter() on page 8-13	Sets the value of a parameter to be passed to the associated stylesheet
setStylesheetURI() on page 8-14	Sets the URI of the stylesheet to be used to process the result.
translateURL() on page 8-14	Returns a string representing an absolute URL resolved relative to the base URI for this request.
useConnectionPooling() on page 8-14	Returns true if connection pooling is desired for this request

Table 8–4 (Cont.) Summary of Methods of XSQLPageRequest

Method	Description
useHTMLErrors() on page 8-15	Returns true if HTML-formatted error messages are desired for this request.
setRequestObject() on page 8-15	Sets a request-scope object.
getRequestObject() on page 8-15	Gets a request-scope object.

createNestedRequest()

Returns an instance of a nested Request.

Syntax

```
public XSQLPageRequest createNestedRequest(
                  java.net.URL pageurl,
                  java.util.Dictionary params);
```

Parameter	Description
pageurl	The URL of the nested request.
params	Optional dictionary of additional parameters.

getConnectionName()

Returns the name of the connection being used for this request May be null if no connection set/in-use.

Syntax

```
public java.lang.String getConnectionName();
```

getErrorWriter()

Returns a PrintWriter to print out errors processing this request.

Syntax

```
public java.io.PrintWriter getErrorWriter();
```

getJDBCConnection()

Gets the JDBC connection being used for this request (can be null).

Syntax

```
public java.sql.Connection getJDBCConnection();
```

getPageEncoding()

Returns encoding of source XSQL Page associated with this request.

Syntax

```
public java.lang.String getPageEncoding();
```

getParameter()

Returns the value of the requested parameter.

Syntax

```
public String getParameter( String name);
```

Parameter	Description
name	The name of the parameter.

getPostedDocument()

Returns the content of Posted XML for this request as an XML Document.

Syntax

```
public oracle.xml.xsql.Document getPostedDocument();
```

getRequestParamsAsXMLDocument()

Returns the content of a Request parameters as an XML Document.

Syntax

```
public oracle.xml.xsql.Document getRequestParamsAsXMLDocument();
```

getRequestType()

Returns a string identifying the type of page request being made.

Syntax

```
public String getRequestType();
```

getSourceDocumentURI()

Returns a String representation of the requested document's URI.

Syntax

```
public String getSourceDocumentURI();
```

getStylesheetParameter()

Gets a stylesheet parameter by name.

Syntax

```
public String getStylesheetParameter( String name);
```

Parameter	Description
name	The stylesheet parameter name.

getStylesheetParameters()

Gets an enumeration of stylesheet parameter names.

Syntax

```
public java.util.Enumeration getStylesheetParameters();
```

getStylesheetURI()

Returns the URI of the stylesheet to be used to process the result.

Syntax

```
public java.lang.String getStylesheetURI();
```

getUserAgent()

Returns a String identifier of the requesting program.

Syntax

```
public java.lang.String getUserAgent();
```

getWriter()

Returns a PrintWriter used for writing out the results of a page request.

Syntax

```
public java.io.PrintWriter getWriter();
```

getXSQLConnection()

Gets the XSQLConnection Object being used for this request Might be null.

Syntax

```
public oracle.xml.xsql.XSQLConnection getXSQLConnection();
```

isIncludedRequest()

Returns true if this request is being included in another.

Syntax

```
public boolean isIncludedRequest();
```

isOracleDriver()

Returns true if the current connection uses the Oracle JDBC Driver.

Syntax

```
public boolean isOracleDriver();
```

printedErrorHeader()

Returns the state of whether an Error Header has been printed.

Syntax

```
public boolean printedErrorHeader();
```

requestProcessed()

Allows Page Request to Perform end-of-request processing.

Syntax

```
public void requestProcessed();
```

setConnectionName()

Sets the connection name to use for this request.

Syntax

```
public void setConnectionName( String connName);
```

Parameter	Description
connName	The name of the connection.

setContentType()

Sets the content type of the resulting page.

Syntax

```
public void setContentType( String mimetype);
```

Parameter	Description
mimetype	The MIME type that describes the content of the resulting page.

setIncludingRequest()

Sets the Including Page Request object for this request.

Syntax

```
public void setIncludingRequest( XMLPageRequest includingEnv);
```

Parameter	Description
includingEnv	The XSQLPageRequest context of the including page.

setPageEncoding()

Sets encoding of source XSQL page associated with this request.

Syntax

```
public void setPageEncoding( String enc);
```

Parameter	Description
enc	The encoding of the current page.

setPageParam()

Sets encoding of source XSQL page associated with this request.

Syntax

```
public void setPageParam( String name, String value);
```

Parameter	Description
name	The name of the page-private parameter.
value	The value of the page-private parameter.

setPostedDocument()

Allows programmatic setting of the Posted Document.

Syntax

```
public void setPostedDocument( org.w3c.dom.Document doc);
```

Parameter	Description
doc	The XML document that has been posted as part of this request.

setPrintedErrorHeader()

Sets whether an Error Header has been printed.

Syntax

```
public void setPrintedErrorHeader( boolean yes);
```

Parameter	Description
yes	Sets whether the error header has been printed.

setStylesheetParameter()

Sets the value of a parameter to be passed to the associated stylesheet.

Syntax

```
public void setStylesheetParameter( java.lang.String name,
                                    java.lang.String value);
```

Parameter	Description
name	The name of the stylesheet parameter.
value	The value of the stylesheet parameter.

setStylesheetURI()

Sets the URI of the stylesheet to be used to process the result.

Syntax

```
public void setStylesheetURI( String uri);
```

Parameter	Description
uri	The uri of the stylesheet to be used to transform this request.

translateURL()

Returns a string representing an absolute URL resolved relative to the base URI for this request.

Syntax

```
public String translateURL( String url);
```

Parameter	Description
url	The url of the stylesheet to be used to translate this request.

useConnectionPooling()

Returns true if connection pooling is desired for this request.

Syntax

```
public boolean useConnectionPooling();
```

useHTMLErrors()

Returns true if HTML-formatted error messages are desired for this request.

Syntax

```
public boolean useHTMLErrors();
```

setRequestObject()

Sets a request-scope object.

Syntax

```
void setRequestObject( String name,
                       Object obj);
```

Parameter	Description
name	Name of the request-scope object to set.
obj	The request-scope object itself.

getRequestObject()

Gets a request-scope object.

Syntax

```
Object getRequestObject( String name);
```

Parameter	Description
name	Name of the request-scope object to retrieve.

XSQLParserHelper Class

Common XML Parsing Routines.

Syntax

```
public final class XSQLParserHelper extends java.lang.Object
```

Table 8–5 Summary of Methods of XSQLParserHelper

Method	Description
XSQLParserHelper() on page 8-16	Class constructor.
newDocument() on page 8-16	Returns a new, empty XML document.
parse() on page 8-16	Parses an XML document from a variety of sources.
parseFromString() on page 8-17	Parses an XML document from a string.
print() on page 8-18	Prints an XML document.

XSQLParserHelper()

Class constructor.

Syntax

```
public  XSQLParserHelper();
```

newDocument()

Returns a new, empty XML document.

Syntax

```
public static oracle.xml.xsql.Document newDocument();
```

parse()

Parses an XML document from a variety of sources. The options are described in the following table.

Syntax	Description
`public static oracle.xml.xsql.Document parse( InputStream is, URL baseUrl, PrintWriter errorWriter);`	Parses an XML document from an InputStream object.
`public static oracle.xml.xsql.Document parse( Reader r, PrintWriter errorWriter);`	Parses an XML document from a Reader object.
`public static oracle.xml.xsql.Document parse( URL url, PrintWriter errorWriter);`	Parses an XML document from a URL.

Parameter	Description
`is`	Input stream containing XML to parse.
`baseUrl`	The base URL of the XML document.
`errorWriter`	PrintWriter to receive any error messages.
`r`	Reader containing XML to parse.
`url`	URL of the XML document to parse.

parseFromString()

Parses an XML document from a string. The options are described in the following table.

Syntax	Description
`public static oracle.xml.xsql.Document parseFromString( StringBuffer xmlString, PrintWriter errorWriter);`	Parses from a StringBuffer.
`public static oracle.xml.xsql.Document parseFromString( String xmlString, PrintWriter errorWriter);`	Parses from a String.

Parameter	Description
xmlString	String containing XML to parser.
errorWriter	PrintWriter to receive any error messages.

print()

Prints an XML document.

Syntax

```
public static void print(  org.w3c.docm.Document doc,
                           java.io.PrintWriter out);
```

Parameter	Description
doc	The XML document to print.
out	The PrintWirter to use for serializing the document.

XSQLRequest Class

Programmatically process a request for an XSQL Page.

Syntax

```
public class XSQLRequest extends java.lang.Object
```

Table 8–6 Summary of Methods of XSQLRequest

Method	Description
XSQLRequest() on page 8-19	Class constructor. Create a Request for an XSQL Page
process() on page 8-19	Process the request, writing output/errors.
processToXML() on page 8-20	Process the request, writing output/errors.
setPostedDocument() on page 8-21	Programmatically set an XML Document to be treated the same as if it were posted as part of the request.

XSQLRequest()

Constructs an instance of `XSQLRequest` from various sources.

Syntax

```
public  XSQLRequest( java.lang.String url);
```

Parameter	Description
`url`	The URL for the XSQL Page source to process.

process()

Process the request, writing output/errors. The options are described in the following table.

Syntax	Description
`public void process();`	Process the request, writing output/errors to System.out/System.err.

Syntax	Description
`public void process( Dictionary params);`	Process the request, writing output/errors to System.out/System.err.
`public void process( Dictionary params, PrintWriter out, PrintWriter err);`	Process the request, writing output/errors to respective PrintWriters.
`public void process( PrintWriter out, PrintWriter err);`	Process the request, writing output/errors to respective PrintWriters.

Parameter	Description
`params`	Dictionary with XSQL Page parameters.
`out`	PrintWriter to use to write the resulting page results.
`err`	PrintWriter to use to write the resulting page errors.

processToXML()

Process the request, writing output/errors. The options are described in the following table.

Syntax	Description
`public org.w3c.dom.Document processToXML()`	Process the request, writing output/errors to System.out/System.err
`public org.w3c.dom.Document processToXML( Dictionary params)`	Process the request, writing output/errors to System.out/System.err
`public org.w3c.dom.Document processToXML( Dictionary params, PrintWriter err)`	Process the request, writing output/errors to respective PrintWriters
`public org.w3c.dom.Document processToXML( PrintWriter err);`	Process the request, writing errors to respective PrintWriters

Parameter	Description
params	Dictionary with XSQL Page parameters
err	PrintWriter to use to write the resulting page errors

setPostedDocument()

Programmatically set an XML Document to be treated the same as if it were posted as part of the request.

Syntax

```
public void setPostedDocument( org.w3c.dom.Document postedDoc)
```

Parameter	Description
postedDoc	DOM Document

XSQLRequestObjectListener Interface

Interface that an object created by an action handler. Can implement to be notified when the current page request processing is completed. Objects that implement this interface and which are added to the current request context using `XSQLPageRequest::setRequestObject()` will be notified when the page processing of the outermost page is completed.

pageProcessingCompleted()

Callback method that allows Request scope objects to be notified when the page processing is complete so they can clean up any allocated resources. Any request-scope object that implements the `XSQLRequestObjectListener` interface will be notified this manner.

Syntax

```
void pageProcessingCompleted();
```

XSQLServletPageRequest Class

Syntax

```
public final class XSQLServletPageRequest extends XSQLPageRequestImpl
```

Table 8–7 Summary of Methods of XSQLServletPageRequest

Method	Description
XSQLServletPageRequest() on page 8-23	Creates a Servlet-specific XSQL Page request.
createNestedRequest() on page 8-24	Returns an instance of a nested Request.
getCookie() on page 8-24	Gets a vale of an HTTP Request cookie.
getHttpServletRequest() on page 8-25	Get the HttpServletRequest that initiated this XSQL Page Request.
getHttpServletResponse() on page 8-25	Get the HttpServletResponse that is associated with this XSQL Page Request.
getParameter() on page 8-25	Use HTTP Parameters as the source of parameters instead.
getPostedDocument() on page 8-25	Retrieves a posted XML document for this request.
getRequestParamsAsXMLDocument() on page 8-25	Retrieves request parameters as an XML document.
getRequestType() on page 8-25	Returns the type of request.
getUserAgent() on page 8-26	Returns the user agent string passed in from the browser.
setContentType() on page 8-26	Sets the content type of the resulting page.
setPageEncoding() on page 8-26	Sets the page encoding.
translateURL() on page 8-26	Returns a string representing an absolute URL resolved relative to the base URI for this request.
useHTMLErrors() on page 8-27	Sets whether HTML-formatted errors should be used.

XSQLServletPageRequest()

Creates a Servlet-specific XSQL Page request.

Syntax

```
public XSQLServletPageRequest(
                    oracle.xml.xsql.HttpServletRequest req,
                    oracle.xml.xsql.HttpServletResponse resp);
```

Parameter	Description
req	Request.
resp	Response.

createNestedRequest()

Returns an instance of a nested Request.

Syntax

```
public XSQLPageRequest createNestedRequest(
                    java.net.URL pageurl,
                    java.util.Dictionary params)
```

Parameter	Description
pageurl	Page URL.
prams	Parameters of the request.

getCookie()

Gets a vale of an HTTP Request cookie.

Syntax

```
public java.lang.String getCookie(   String name);
```

Parameter	Description
name	Name of the cookie to get.

getHttpServletRequest()

Get the HttpServletRequest that initiated this XSQL Page Request.

Syntax

```
public oracle.xml.xsql.HttpServletRequest getHttpServletRequest();
```

getHttpServletResponse()

Get the HttpServletResponse that is associated with this XSQL Page Request.

Syntax

```
public oracle.xml.xsql.HttpServletResponse getHttpServletResponse();
```

getParameter()

Use HTTP Parameters as the source of parameters instead.

Syntax

```
public java.lang.String getParameter( String name);
```

Parameter	Description
`name`	Name of the paramete4r whose value is to be retrieved.

getPostedDocument()

Retrieves a posted XML document for this request. Returns `NULL` if there is none.

Syntax

```
public oracle.xml.xsql.Document getPostedDocument();
```

getRequestParamsAsXMLDocument()

Retrieves request parameters as an XML document.

Syntax

```
public oracle.xml.xsql.Document getRequestParamsAsXMLDocument();
```

getRequestType()

Returns the type of request (“Servlet”, “Programmatic”, “Command Line”, or “Custom”).

Syntax

```
public java.lang.String getRequestType();
```

getUserAgent()

Returns the user agent string passed in from the browser.

Syntax

```
public java.lang.String getUserAgent();
```

setContentType()

Sets the content type of the resulting page.

Syntax

```
public void setContentType( String mimetype);
```

Parameter	Description
mimtype	The MIME type that describes the content of the resulting page.

setPageEncoding()

Sets the page encoding.

Syntax

```
public void setPageEncoding( String enc);
```

Parameter	Description
enc	The page encoding.

translateURL()

Returns a string representing an absolute URL resolved relative to the base URI for this request.

Syntax

```
public java.lang.String translateURL( String path);
```

Parameter	Description
path	The relative URL which will be translated to an absolute URL.

useHTMLErrors()

Sets whether HTML-formatted errors should be used.

Syntax

```
public boolean useHTMLErrors();
```

XSQLStylesheetProcessor Class

XSLT Stylesheet Processing Engine.

Syntax

```
public final class XSQLStylesheetProcessor extends java.lang.Object
```

Table 8–8 Summary of Methods of XSQLStylesheetProcessor

Method	Description
XSQLStylesheetProcessor() on page 8-28	Processes CSLT stylesheet transformations.
processToDocument() on page 8-28	Transforms an XML document by an XSLT stylesheet and returns the result as an XML document.
processToWriter() on page 8-29	Transforms an CML Document by an XSLT stylesheet and writes the result to a PrintWriter.
getServletContext() on page 8-29	Gets the Http Servlet Context.

XSQLStylesheetProcessor()

Processes CSLT stylesheet transformations.

Syntax

```
public  XSQLStylesheetProcessor();
```

processToDocument()

Transforms an XML document by an XSLT stylesheet and returns the result as an XML document.

Syntax

```
public static oracle.xml.xsql.Document processToDocument(
                org.w3c.dom.Document xml,  String xslURI, XSQLPageResuest env);
```

Parameter	Description
`xml`	The XML document.
`xslURI`	The XSL stylesheet URI.

Parameter	Description
env	The XSQLPageRequest context.

processToWriter()

Transforms an CML Document by an XSLT stylesheet and writes the result to a PrintWriter.

Syntax

```
public static void processToWriter(
                    oracle.xml.xsql.Document xml,
                    String xslURI,
                    XSQLPageResuest env);
```

Parameter	Description
xml	The XML document.
xslURI	The XSL stylesheet URI.
env	The XSQLPageRequest context.

getServletContext()

Gets the Http Servlet Context.

Syntax

```
javax.servlet.ServletContext getServletContext();
```

XSQLConnectionManager Interface

One of two interfaces that must be implemented to override the built-in connection manager implementation. The XSQL Page Processor asks the `XSQLConnectionManagerFactory` associated with each request to `create()` an instance of an XSQLConnectionManager to service the current request.

In multithreaded environments, the implementation of XSQLConnectionManager must insure that an XSQLConnection instance returned by `getConnection()` is not used by another thread until it has been released by the XSQL Page Processor after the call to `releaseConnection()`.

Syntax

```
public interface XSQLConnectionManager;
```

Table 8–9 Summary of Methods of XSQLConnectionManager

Method	Description
getConnection() on page 8-30	Gets a connection from the connection manager.
releaseConnection() on page 8-31	Releases the connection.

getConnection()

Gets a connection from the connection manager.

Syntax

```
XSQLConnection getConnection( String connName,
                              XSQLPageRequest env);
```

Parameter	Description
`connName`	The connection name.
`env`	The XSQLPageRequest context.

releaseConnection()

Releases the connection.

Syntax

```
void releaseConnection( XSQLConnection theConn,
                        XSQLPageRequest env);
```

Parameter	Description
theConn	The XSQL Connection object to be released.
env	The XSQLPageRequest context.

XSQLConnectionManagerFactory Interface

One of two interfaces that must be implemented to override the built-in connection manager implementation. The XSQL Page Processor asks the `XSQLConnectionManagerFactory` associated with each request to `create()` an instance of an `XSQLConnectionManager` to service the current request.

Syntax

```
public interface XSQLConnectionManagerFactory
```

create()

Returns an instance of an XSQLConnectionManager. Implementation can decide whether it is a new XSQLConnectionManager or a shared singleton.

Syntax

```
XSQLConnectionManager create();
```

XSQLDocumentSerializer Interface

Interface that must be implemented by all XSQL Serializers which serialize an XSQL data page as an XML Document to a PrintWriter.

Upon encountering a serializer="XXX" pseudo-attribute in an `<?xml-stylesheet?>` processing instruction, the XSQL Page Processor invokes the associated serializer by:

- Constructing an instance of the serializer using the no-args constructor
- Invoking the XSQL document serializer's `serialize()` method

An implementation of XSQLDocumentSerializer is expected to do the following actions.

- First, call `env.setContentType()` to set the content type
- Then, call `env.getWriter()` to get the Writer to write to

If the serializer throws an unhandled exception, the XSQL Page Processor will format the stacktrace.

See oracle.xml.xsql.src.serializers.XSQLSampleSerializer for an example.

Syntaxr

```
public interface XSQLDocumentSerializer
```

serialize()

Serializes the resulting XML document for the output writer.

Syntax

```
void serialize( org.w3c.dom.Document doc,
                XSQLPageRequest env);
```

Parameter	Description
doc	The XML document to serialize.
env	The XSQLPageRequest context.

9

TransX Utility for Java

The TransX Utility simplifies the loading of translated seed data and messages into a database. It also reduces internationalization costs by preparing strings for translation, translating the strings, and leading. The TransX Utility minimizes translation data format errors and it accurately loads the translation contents into pre-determined locations in the database.

Development groups that load translated messages and seed data can use the TransX Utility to simplify meeting internationalization requirements. Once the data is in a predefined format, the TransX Utility validates its format. Loading translated data is automated because the encoding of the files takes advantage of XML which *describes* the encoding. This means that possible loading errors due to incorrect encoding are eliminated as long as the data file conforms to the XML standard.

This chapter contains the following sections:

- TransX Utility Command Line Interface
- TransX Utility Application Programming Interface

See Also:

- *Oracle Application Developer's Guide - XML*

TransX Utility Command Line Interface

Syntax

```
java oracle.xml.transx.loader [options] connect_string username password
datasource [datasource(s)]
java oracle.xml.transx.loader -v datasource [datasource(s)]
java oracle.xml.transx.loader -x connect_string username password table
[column(s)]
java oracle.xml.transx.loader -s connect_string username password filename table
[column(s)]
```

Table 9–1 Command Line Parameters of TransX Utility

Parameter	Description
connect_ string	JDBC connect string. The connect string information can be omitted through the @ symbol. jdbc:oracle:thin:@ will be supplied.
username	Database user name.
password	Password for the database user.
datasource	An XML data source.
option	One of the options in Table 9–2, " TransX Utility Command Line Options".

Table 9–2 TransX Utility Command Line Options

Option	Description
-u	Update existing rows. When this option is specified, existing rows are not skipped but updated. To exclude a column from the update operation, specify the useforupdate attribute to be no.
-e	Raise exception if a row is already in the database. When this option is specified, an exception will be thrown if a duplicate row is found. By default, duplicate rows are skipped. Rows are considered duplicate if the values for lookup-key column(s) in the database and the dataset are the same.
-x	Print data in the database in predefined format. Causes TransX to perform the unloading, but unlike the -s option, it prints the output to stdout. Because intervention of the operating system may result in data loss due to unexpected transcoding, redirecting this output to a file is discouraged.

Table 9–2 (Cont.) TransX Utility Command Line Options

Option	Description
-s	Save data in the database into a file with predefined format. This is an option to perform unloading. It queries the database, formats the result into the predefined XML format and stores it under the specified file name.
-p	Print the XML to load. Prints out the dataset for insert in the canonical format of XSU.
-t	Print the XML for update. Prints out the dataset for update in the canonical format of XSU.
-o	Omit validation (by default, the dataset is validated while it is parsed). Causes TransX to skip the format validation, which is performed by default.
-v	Validate the data format and exit without loading. Causes TransX to perform validation and exit.
-w	Preserve white space. Causes TransX to treat whitespace characters (such as \t, \r, \n, and") as significant. By default, consecutive whitespace characters in string data elements are condensed into one space character.

Table 9–3 TransX Utility Command Line Exceptions

Exception	Description
-u , -e	Mutually exclusive
-v	Must be the only option followed by data
-x	Must be the only option followed by connect info and SQL query
	Omitting all arguments will result in the display of the front-end usage information shown in the table.

TransX Utility Application Programming Interface

This section describes the TransX Utility application programming interface, and consists of the following classes:

- loader Class
- TransX Interface

loader Class

Provides the methods to instantiate the TransX Interface, through which loading operations are performed.

Syntax

```
oracle.xml.transx.loader
```

Table 9–4 Summary of Methods of loader

Method	Description
getLoader() on page 9-5	Get an instance of TransX.
main() on page 9-5	The command-line interface.

getLoader()

Get an instance of TransX. See also TransX Interface.

Syntax

```
public static TransX getLoader();
```

main()

The command-line interface main function.

Syntax

```
public static void main( String[] args);
```

Parameter	Description
`args`	Command line parameters.

TransX Interface

Data Loading Tool API.

Syntax

```
public interface TransX
```

Table 9–5 Summary of Methods of TransX

Method	Description
close() on page 9-6	Help reclaim used resources.
load() on page 9-6	Load a dataset on a file or URL.
open() on page 9-7	Start a data loading session.
setLoadingMode() on page 9-7	Set the operation mode on duplicates.
setPreserveWhitespace() on page 9-8	Tell the loader to treat white spaces as significant.
setValidationMode() on page 9-8	Set the validation mode.
unload() on page 9-9	Unload a dataset.
validate() on page 9-9	Validate a dataset on a file or URL.

close()

Helps reclaim used resources. This method should be called after loading is complete. See also `open()`.

Reports a `java.sql.SQLException`.

Syntax

```
public void close();
```

load()

Loads a dataset on a file. A database connection must have been established by a preceding call to `open()`. Returns the total number of inserted or updated row elements. Reports a `java.lang.Exception`. See also `open()`. The options are listed in the following table.

Syntax	Description
`public int load( String file);`	Loads the XML file specified by the `file` parameter.
`public int load( URL url);`	Loads the XML document to which the `url` refers.

Parameter	Description
`file`	file name
`url`	URL string

open()

Starts a data loading session. This method establishes a database connection to the given JDBC URL to be used for subsequent load() call(s).

Reports a `java.sql.SQLException`. See also `load()`.

Syntax

```
public void open( String constr,
                  String user,
                  String pwd);
```

Parameter	Description
`constr`	A database url of the form: `jdbc:Oracle:<driver_type>:@[additional_parameters]`
`user`	The database user on whose behalf the connection is being made.
`pwd`	The user's password.

setLoadingMode()

Sets the operation mode on duplicates. In the case when there are one or more existing rows in the database whose values in the key columns are the same as those in the dataset to be loaded, the loader, depending on the current loading mode specified by this method,

- skips duplicate rows (default),
- updates them, or
- reports an exception.

Syntax

```
public void setLoadingMode(int mode);
```

Parameter	Description
mode	A loading mode. The following constants are defined in Loading Mode class: SKIP_DUPLICATES (default) UPDATE_DUPLICATES EXCEPTION_ON_DUPLICATES

setPreserveWhitespace()

Instructs the loader to treat white spaces as significant. By default, the flag is FALSE. If this call is not made, or if FALSE is passed to the flag argument, the loader ignores the type of white space characters in the dataset and loads them as space characters. Consecutive white space characters in the dataset are treated as one space character.

Syntax

```
public void setPreserveWhitespace( boolean flag);
```

Parameter	Description
flag	TRUE for exceptions, otherwise FALSE.

setValidationMode()

Sets the validation mode. The flag is set to TRUE by default: If this call is not made or if true is passed to the flag argument, the loader performs validation of the dataset format against the canonical schema definition on each load() call. The validation mode should be disabled only if the dataset has already been validated. See also validate().

Syntax

```
public void setValidationMode(boolean flag);
```

Parameter	Description
flag	Determines whether the loader should be validating.

unload()

Unloads a dataset. The options are listed in the following table. Returns the dataset in XML. Reports a `java.lang.Exception`. The options are listed in the following table.

Syntax	Description
`public java.io.Reader unload( String table, String[] columns )*;`	Returns the Reader which can be read.
`public void unload( String table, String[] columns, Writer out )*;`	Writes the unloaded dataset into the given writer.

Parameter	Description
table	A table name.
columns	Column names, means * when null.
out	A Writer to write to.

validate()

Validate a dataset on a file or URL. Once validated, the validation mode can safely be disabled. The instance document does not have to be opened for validation. See also `setValidationMode(). Returns TRUE` if successfully validated.Reports an `oracle.xml.parser.schema.XSDException.`

Syntax

```
public boolean validate( String datasrc);
```

Parameter	Description
datasrc	A file name or URL string

10

Oracle XML JavaBeans

Oracle XML JavaBeans are synonymous with the following packages:

- oracle.xml.async Package
- oracle.xml.dbviewer Package
- oracle.xml.srcviewer Package
- oracle.xml.transviewer Package
- oracle.xml.treeviewer Package
- oracle.xml.differ Package

See Also:

- *Oracle Application Developer's Guide - XML*

oracle.xml.async Package

This is a non-visual bean. It enables asynchronous DOM parsing in separate threads in the background. It utilizes the EventHandler interface to notify the calling class when the job is complete. The classes of the `oracle.xml.async` are summarized in Table 10–1.

Table 10–1 Summary of Classes of oracle.xml.async

Class	Description
DOMBuilder Class on page 10-3	This class encapsulates an eXtensible Markup Language (XML) 1.0 parser to parse an XML document and build a DOM tree.
DOMBuilderBeanInfo Class on page 10-13	This class provides information about the DOMBuilder Bean.
DOMBuilderErrorEvent Class on page 10-15	This class defines the error event which is sent when parse exception occurs.
DOMBuilderErrorListener Interface on page 17	This interface must be implemented in order to receive notifications when error is found during parsing.
DOMBuilderEvent Class on page 10-18	The event object that DOMBuilder uses to notify all registered listeners about parse events.
DOMBuilderListener Interface on page 10-20	This interface must be implemented in order to receive notifications about events during the asyncronous parsing.
ResourceManager Class on page 10-22	This class implements a simple semaphore that maintains access to a fixed number of logical resources.
XSLTransformer Class on page 24	Applies XSL transformation in a background thread.
XSLTransformerBeanInfo Class on page 10-30	This class provides information about the XSLTransformer Bean.
XSLTransformerErrorEvent Class on page 10-32	The error event object that XSLTransformer uses to notify all registered listeners about transformation error events.

DOMBuilder Class

This class encapsulates an eXtensible Markup Language (XML) 1.0 parser to parse an XML document and build a DOM tree. The parsing is done in a separate thread and DOMBuilderListener interface must be used for notification when the tree is built.

Syntax

```
public class DOMBuilder extends java.lang.Object implements
java.io.Serializable, oracle.xml.async.DOMBuilderConstants, java.lang.Runnable
```

Table 10–2 Fields of DOMBuilder

Field	Syntax	Description
inSource	protected org.xml.sax.InputSource inSource	InputSource containing XML data to parse
url	protected java.net.URL url	URL to parse XML data from
inStream	protected java.io.InputStream inStream	InputStream containing XML data to parse
inString	protected java.lang.String inString	String containing the URL to parse XML data from
methodToCall	protected int methodToCall	XML Parser method to call based on input types
reader	protected java.io.Reader reader	java.io.Reader containing XML data to be parsed
result	protected oracle.xml.async.XMLDocument result	XML Document being parsed
rootName	protected java.lang.String rootName	Name of the XML element to be treated as root

Table 10–3 Summary of Methods of DOMBuilder

Method	**Description**
DOMBuilder() on page 10-5	Creates a new parser object.
addDOMBuilderErrorListener() on page 10-5	Adds DOMBuilderErrorListener.
addDOMBuilderListener() on page 10-5	Adds DOMBuilderListener.
getDoctype() on page 10-6	Gets the DTD.
getDocument() on page 10-6	Gets the document for parsing.
getId() on page 10-6	Returns the parser object ID.
getReleaseVersion() on page 10-6	Returns the release version of Oracle XML Parser.
getResult() on page 10-6	Gets the document being parsed.
getValidationMode() on page 10-6	Returns the validation mode.
parse() on page 10-6	Parses the XML from given input.
parseDTD() on page 10-7	Parses the XML External DTD.
removeDOMBuilderErrorListener() on page 10-8	Removes the DOMBuilderErrorListener.
removeDOMBuilderListener() on page 10-9	Removes the DOMBuilderListener.
run() on page 10-9	Runs in a thread.
setBaseURL() on page 10-9	Sets the base URL for lading external entities and DTD.
setDebugMode() on page 10-9	Sets a flag to run on debug information in the document.
setDoctype() on page 10-10	Sets the DTD.
setErrorStream() on page 10-10	Creates an output stream for errors and warnings.
setNodeFactory() on page 10-11	Sets the node factory.
setPreserveWhitespace() on page 10-11	Sets the white space preservation mode.
setValidationMode() on page 10-11	Sets the validation mode.
showWarnings() on page 10-12	Determines whether to print warnings.

DOMBuilder()

Creates a new parser object. The options are described in the following table.

Syntax	Description
`public DOMBuilder();`	Creates a new parser object.
`public DOMBuilder( int id);`	Creates a new parser object with a given id.

Parameter	Description
`id`	The DOMBuilder id.

addDOMBuilderErrorListener()

Adds DOMBuilderErrorListener.

Syntax

```
public void addDOMBuilderErrorListener( DOMBuilderErrorListener p0);
```

Parameter	Description
`p0`	The DOMBuilderListener to add.

addDOMBuilderListener()

Adds DOMBuilderListener

Syntax

```
public void addDOMBuilderListener(DOMBuilderListener p0);
```

Parameter	Description
`p0`	The DOMBuilderListener to add.

getDoctype()

Gets the DTD.

Syntax

```
public synchronized oracle.xml.async.DTD getDoctype();
```

getDocument()

Gets the document for parsing.

Syntax

```
public synchronized oracle.xml.async.XMLDocument getDocument();
```

getId()

Returns the parser object id (DOMBuilder).

Syntax

```
public int getId();
```

getReleaseVersion()

Returns the release version of the Oracle XML Parser, as a String.

Syntax

```
public synchronized java.lang.String getReleaseVersion();
```

getResult()

Gets the document being parsed.

Syntax

```
public synchronized org.w3c.dom.Document getResult();
```

getValidationMode()

Returns the validation mode; `TRUE` if XML parser is validating, `FALSE` otherwise.

Syntax

```
public synchronized boolean getValidationMode()
```

parse()

Parses the XML from given input. Throws the following exceptions:

- `XMLParseException` If syntax or other error encountered.

- `SAXException` Any SAX exception, possibly wrapping another exception.
- `IOExceptiopn` I/O Error.

Syntax	Description
`public final synchronized void parse ( InputSource in);`	Parses the XML from an InputSource
`public final synchronized void parse ( InputStream in);`	Parses the XML from an InputStream; the base URL should be set for resolving external entities and DTD.
`public final synchronized void parse ( Reader r);`	Parses the XML from an Reader; the base URL should be set for resolving external entities and DTD.
`public final synchronized void parse ( String urlName);`	Parses the XML from the URL indicated by the argument.
`public final synchronized void parse ( URL url);`	Parses the XML document pointed to by the given URL and creates the corresponding XML document hierarchy.

Parameter	Description
`in`	The input to parse.
`r`	The Reader containing XML data to parse.
`urlName`	The String containing the URL from which to parse.
`url`	The ULR that points to the XML document to parse.

parseDTD()

Parses the XML External DTD. Throws the following exceptions:

- `XMLParseException` If syntax or other error encountered.
- `SAXException` Any SAX exception, possibly wrapping another exception.
- `IOExceptiopn` I/O Error.

Syntax	Description
`public final synchronized void parseDTD ( InputSource in, String rootName);`	Parses from a given InputSource.
`public final synchronized void parseDTD ( InputStream in, String rootName);`	Parses from a givenInputStream. The base URL should be set for resolving external entities and DTD.
`public final synchronized void parseDTD ( Reader in, String rootName);`	Parses from a given Reader. The base URL should be set for resolving external entities and DTD.
`public final synchronized void parseDTD ( String urlName, String rootName);`	Parses from the URL indicated.
`public final synchronized void parseDTD ( URL url, String rootName);`	Parses the XML External DTD document pointed to by the given URL and creates the corresponding XML document hierarchy.

Parameter	Description
`in`	The input to parse.
`rootName`	The element to be used as root element.
`r`	The Reader containing XML data to parse.
`urlName`	The String containing the URL from which to parse.
`utl`	The ULR that points to the XML document to parse.

removeDOMBuilderErrorListener()

Removes DOMBuilderErrorListener.

Syntax

```
public synchronized void removeDOMBuilderErrorListener(
                DOMBuilderErrorListener p0);
```

Parameter	Description
po	The DOMBuilderErrorListener to remove.

removeDOMBuilderListener()

Removes DOMBuilderListener.

Syntax

```
public synchronized void removeDOMBuilderListener(
                DOMBuilderListener p0);
```

Parameter	Description
p1	The DOMBuilderListener to remove.

run()

This method runs in a thread. It is specified in java.lang.Runnable.run() in interface java.lang.Runnable.

Syntax

```
public void run();
```

setBaseURL()

Sets the base URL for loading external entities and DTDs. This method should to be called if the parse(InputStream) option is used to parse the XML Document.

Syntax

```
public synchronized void setBaseURL( java.net.URL url);
```

Parameter	Description
url	The base URL.

setDebugMode()

Sets a flag to turn on debug information in the document.

Syntax

```
public void setDebugMode(boolean flag);
```

Parameter	Description
flag	Determines whether debug information is stored; TRUE when stored.

setDoctype()

Sets the DTD.

Syntax

```
public synchronized void setDoctype(oracle.xml.async.DTD dtd)
```

Parameter	Description
dtd	The DTD to set and use while parsing.

setErrorStream()

Creates an output stream for errors and warnings. If an output stream for errors is not specified, the parser will use the standard error output stream System.err for outputting errors and warnings. The options are described in the following table.

Syntax	Description
`public final synchronized void setErrorStream( OutputStream out);`	Uses OutputSteam.
`public final synchronized void setErrorStream( OutputStream out, String enc);`	Uses OutputSteam. An IOException is thrown if the encoding specified is not supported.
`public final synchronized void setErrorStream( PrintWriter out);`	Uses PrintWriter.

Parameter	Description
out	The the output for errors and warnings.
enc	The encoding to use.

setNodeFactory()

Sets the node factory. Applications can extend the NodeFactory and register it through this method. The parser will then use the user supplied NodeFactory to create nodes of the DOM tree. Throws `XMLParseException` when using an invalid factory.

Syntax

```
public synchronized void setNodeFactory(
                oracle.xml.async.NodeFactory factory);
```

Parameter	Description
factory	The NodeFactory to set.

setPreserveWhitespace()

Sets the white space preservation mode.

Syntax

```
public synchronized void setPreserveWhitespace( boolean flag);
```

Parameter	Description
flag	The preserving mode; `TRUE` to preserve whitespace, `FALSE` otherwise.

setValidationMode()

Sets the validation mode.

Syntax

```
public synchronized void setValidationMode(boolean yes);
```

Parameter	Description
yes	Determines whether XML parser should be validating; TRUE for validation.

showWarnings()

Determines whether to print warnings.

Syntax

```
public synchronized void showWarnings(boolean yes);
```

Parameter	Description
yes	Switch; TRUE to print warnings, FALSE otherwise.

DOMBuilderBeanInfo Class

This class provides information about the DOMBuilder Bean.

Syntax

```
public class DOMBuilderBeanInfo extends java.beans.SimpleBeanInfo
```

Table 10–4 Summary of Methods of DOMBuilderBeanInfo

Method	Description
DOMBuilderBeanInfo() on page 10-13	The default constructor.
getIcon() on page 10-13	Gets an image object that can be used to represent the DOMBuilder bean in toolbars, and so on.
getPropertyDescriptors() on page 10-14	Retrieves the array of DOMBuilder bean's editable PropertyDescriptiors.

DOMBuilderBeanInfo()

The default constructor.

Syntax

```
public  DOMBuilderBeanInfo();
```

getIcon()

Gets an image object that can be used to represent the DOMBuilder bean in toolbars, toolboxes, and so on. Returns an image object representing the requested icon type.

Syntax

```
public java.awt.Image getIcon( int iconKind);
```

Parameter	Description
iconKind	The kind of icon requested.

getPropertyDescriptors()

Retrieves the array of DOMBuilder bean's editable PropertyDescriptiors.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

DOMBuilderErrorEvent Class

This class defines the error event which is sent when parse exception occurs.

Syntax

```
public class DOMBuilderErrorEvent extends java.util.EventObject
```

Table 10–5 Fields of DOMBuilderErrorEvent

Field	Syntax	Description
e	protected java.lang.Exception	The exception being raised.

DOMBuilderErrorEvent()

Constructor for DOMBuilderErrorEvent.

Syntax

```
public  DOMBuilderErrorEvent( Object p0,
                              Exception e);
```

Parameter	Description
p0	The Object that created this error event.
e	The Exception being raised.

getException()

Retrieves the exception being raised.

Syntax

```
public java.lang.Exception getException();
```

getMessage()

Returns the error message generated by the parser, as a String.

Syntax

```
public java.lang.String getMessage();
```

DOMBuilderErrorListener Interface

This interface must be implemented in order to receive notifications when error is found during parsing. The class implementing this interface must be added to the DOMBuilder using addDOMBuilderErrorListener method.

Syntax

```
public interface DOMBuilderErrorListener extends java.util.EventListener
```

domBuilderErrorCalled()

This method is called when a parse error occurs.

Syntax

```
public void domBuilderErrorCalled( DOMBuilderErrorEvent p0);
```

Parameter	Description
p0	The DOMBuilderErrorEvent object produced by the DOMBuilder as result of parsing error.

DOMBuilderEvent Class

The event object that DOMBuilder uses to notify all registered listeners about parse events.

Syntax

```
public class DOMBuilderEvent extends java.util.EventObject
```

Table 10–6 Fields of DOMBuilderEvent

Field	Syntax	Description
id	protected int id	ID of the source DOMBuilder object

Table 10–7 Summary of Methods of DOMBuilderEvent

Method	Description
DOMBuilderEvent() on page 10-18	Creates a new DOMBuilderEvent.
getID() on page 10-19	Returns unique id of the source DOMBuilder for the event.

DOMBuilderEvent()

Creates a new DOMBuilderEvent.

Syntax

```
public  DOMBuilderEvent( Object p0, int p1);
```

Parameter	Description
p0	The Object creating this event.
p1	Id of the DOMBuilder creating this event.

getID()

Returns unique id of the source DOMBuilder for this event, which can be used to identify which instance of the DOMBuilder generated this event in cases where multiple instances of DOMBuilder may be working in background.

Syntax

```
public int getID();
```

DOMBuilderListener Interface

This interface must be implemented in order to receive notifications about events during the asyncronous parsing. The class implementing this interface must be added to the DOMBuilder using addDOMBuilderListener method.

Syntax

```
public interface DOMBuilderListener extends java.util.EventListener
```

Table 10–8 Summary of Methods of DOMBuilderListener

Method	Description
domBuilderError() on page 10-17	This method is called when parse error occur.
domBuilderOver() on page 10-20	This method is called when the parse is complete.
domBuilderStarted() on page 10-21	This method is called when parse starts

domBuilderError()

This method is called when parse error occur.

Syntax

```
public void domBuilderError( DOMBuilderEvent p0);
```

Parameter	Description
p0	The DOMBuilderEvent object produced by the DOMBuilder.

domBuilderOver()

This method is called when the parse is complete.

Syntax

```
public void domBuilderOver( DOMBuilderEvent p0);
```

Parameter	Description
p0	The DOMBuilderEvent object produced by the DOMBuilder.

domBuilderStarted()

This method is called when parse starts.

Syntax

```
public void domBuilderStarted( DOMBuilderEvent p0);
```

Parameter	Description
p0	The DOMBuilderEvent object produced by the DOMBuilder.

ResourceManager Class

Implements a semaphore and maintains access to a fixed number of logical resources.

Syntax

```
public class ResourceManager extends java.lang.Object
```

Table 10–9 Summary of Methods of ResourceManager

Method	Description
ResourceManager() on page 10-22	The ResourceManager constructor.
activeFound() on page 10-23	Checks if any of the logical resources being managed are in active use.
getResource() on page 10-23	If the number of resources available for use is nonzero, decreases the number of resources by one. Otherwise, waits until a resource is released and it becomes available for use.
releaseResource() on page 10-23	Releases a single resource, increasing the number of resources available.
sleep() on page 10-23	Allows use of Thread.sleep() without try/catch.

ResourceManager()

The ResourceManager constructor.

Syntax

```
public  ResourceManager(int i);
```

Parameter	Description
i	The number of resources to manage.

activeFound()

Checks if any of the logical resources being managed are in active use. Returns `TRUE` if one or more resources in use, `FALSE` if no resources in use.

Syntax

```
public boolean activeFound();
```

getResource()

If the number of resources available for use is nonzero, the method decreases the number of resources by one. Otherwise, it waits until a resource is released and it becomes available for use.

Syntax

```
public synchronized void getResource();
```

releaseResource()

Releases a resource. When this method is called, the number of resources available is increased by one.

Syntax

```
public void releaseResource();
```

sleep()

Allows use of Thread.sleep() without try/catch.

Syntax

```
public void sleep(int i);
```

Parameter	Description
i	The number of resources to manage.

XSLTransformer Class

Applies XSL transformation in a background thread.

Syntax

```
public class XSLTransformer extends java.lang.Object implements
     java.io.Serializable, oracle.xml.async.XSLTransformerConstants,
     java.lang.Runnable

java.lang.Object
```

Table 10–10 Fields of XSLTransformer

Field	Syntax	Description
`methodToCall`	`protected int methodToCall`	The XSL transformation method to call based on input types.
`result`	`protected oracle.xml.async.DocumentFragment result`	Transformation result document.

Table 10–11 Summary of Methods of XSLTransformer

Method	Description
XSLTransformer() on page 10-25	XSLTransformer class constructor.
addXSLTransformerErrorListener() on page 10-25	Adds and XSLTransformerErrorListener.
addXSLTransformerListener() on page 10-26	Adds and XSLTransformerListener.
getId() on page 10-26	Returns a unique XSLTransformer id.
getResult() on page 10-26	Returns the document fragment of the XSL transformation for the resulting document.
processXSL() on page 10-26	Initiates XSL Transformation in the background. The control is returned immediately.
removeDOMTransformerErrorListener() on page 10-27	Removes an XSLTransformerErrorListener.

Table 10–11 (Cont.) Summary of Methods of XSLTransformer

Method	Description
removeXSLTransformerListener() on page 10-28	Removes an XSLTransformerListener.
run() on page 10-28	Starts a separate thread to perform the XSLTransformation.
setErrorStream() on page 10-28	Sets the error stream to be used by XML processor.
showWarnings() on page 10-28	Sets the showWarnings flag used by the XSL processor.

XSLTransformer()

XSLTransformer constructor. The options are described in the following table.

Syntax	Description
`public XSLTransformer();`	XSLTransformer constructor.
`public XSLTransformer( int id);`	XSLTransformer constructor accepting an identifier.

Parameter	Description
`id`	Integer used to identify the XSLTransformer instance during event processing

addXSLTransformerErrorListener()

Adds an XSLTransformerErrortListener.

Syntax

```
public void addXSLTransformerErrorListener(
                XSLTransformerErrorListerner p0);
```

Parameter	Description
`p0`	XSLTransformerErrorListener to be added.

addXSLTransformerListener()

Adds a XSLTransformerListener.

Syntax

```
public void addXSLTransformerListener( XSLTransformerListerner p0);
```

Parameter	Description
p0	XSLTransformerListener to be added.

getId()

Returns the unique XSLTransformer id.

Syntax

```
public int getId();
```

getResult()

Returns the document fragment of the XSL transformation for the resulting document. Called only after receiving notification that the transformation is complete. Since the transformation occurs in background and asyncronously, calling this method immediately after `processXSL()` will result in holding the control until the result is available.

Syntax

```
public synchronized oracle.xml.async.DocumentFragment getResult();
```

processXSL()

Initiates XSL Transformation in the background. The control is returned immediately. An `XSLException` is thrown if an error occurs during XSL transformation. The options are described in the following table.

Syntax	Description

`public void processXSL(`
`  oracle.xml.async.XSLStylesheet xsl,`
`  InputStream xml,`
`  URL ref)`
	The source XML document is provided as an InputStream.

Syntax	Description
`public void processXSL( oracle.xml.async.XSLStylesheet xsl, j Reader xml, URL ref);`	The source XML document is provided as a Reader.
`public void processXSL( oracle.xml.async.XSLStylesheet xsl, URL xml, URL ref);`	The source XML document is provided through a URL.
`public void processXSL( oracle.xml.async.XSLStylesheet xsl, oracle.xml.async.XMLDocument xml);`	The source XML document is provided as a DOM tree.
`public void processXSL( oracle.xml.async.XSLStylesheet xsl, oracle.xml.async.XMLDocument xml, j OutputStream os);`	The source XML document is provided as a DOM tree, and the output is written into an OutputStream.

Parameter	Description
`xsl`	The stylesheet to be used for XSL transformation
`xml`	The XML document to be used
`ref`	The reference URL to resolve external entities in input XML
`os`	Output to which the XSL transformation result is written

removeDOMTransformerErrorListener()

Removes an XSLTransformerErrorListener.

Syntax

```
public synchronized void removeDOMTransformerErrorListener(
                XSLTransformerErrorListener p0);
```

Parameter	Description
`p0`	The XSLTransformerErrorListener to be removed.

removeXSLTransformerListener()

Removes a XSLTransformerListener.

Syntax

```
public synchronized void removeXSLTransformerListener(
                XSLTransformerListener p0);
```

Parameter	Description
p0	The XSLTransformerListener to be removed.

run()

Starts a separate thread to perform the XSLTransformation. Specified by java.lang.Runnable.run() in java.lang.Runnable Interface.

Syntax

```
public void run();
```

setErrorStream()

Sets the error stream used by the XSL processor.

Syntax

```
public final void setErrorStream( java.io.OutputStream out);
```

Parameter	Description
out	The error output stream for the XSL processor.

showWarnings()

Sets the showWarnings flag used by the XSL processor.

Syntax

```
public final void showWarnings( boolean yes);
```

Parameter	Description
yes	Indicates whether XSL processor warnings should be shown; TRUE to show warnings, FALSE otherwise.

XSLTransformerBeanInfo Class

This class provides information about the XSLTransformer Bean.

Syntax

```
public class XSLTransformerBeanInfo extends java.beans.SimpleBeanInfo
```

Table 10–12 Summary of Methods of XSLTransformerBeanInfo

Method	Description
XSLTransformerBeanInfo() on page 10-30	The default constructor.
getIcon() on page 10-30	Retrieves an image representing the requested icon type.
getPropertyDescriptors() on page 10-31	Retrieves the array of XSLTransformer bean's editable PropertyDescriptiors.

XSLTransformerBeanInfo()

The default Constructor.

Syntax

```
public  XSLTransformerBeanInfo();
```

getIcon()

Retrieves an image object representing the requested icon type for XSLTransformer bean in toolbars, toolboxes, and so on. Overrides `getIcon()` in `java.beans.SimpleBeanInfo` class.

Syntax

```
public java.awt.Image getIcon( int iconKind);
```

Parameter	Description
`iconKind`	The kind of icon requested.

getPropertyDescriptors()

Retrieves the array of XSLTransformer bean's editable PropertyDescriptiors. Overrides `getPropertyDescriptors()` in `java.beans.SimpleBeanInfo` class.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

XSLTransformerErrorEvent Class

The error event object that XSLTransformer uses to notify all registered listeners about transformation error events.

Syntax

```
public class XSLTransformerErrorEvent extends java.util.EventObject
```

Table 10–13 Fields of XSLTransformerErrorEvent

Field	Syntax	Description
e	protected java.lang.Exception e	The exception being raised.

Table 10–14 Summary of Methods of XSLTransformerErrorEvent

Method	Description
XSLTransformerErrorEvent() on page 10-32	Constructor for XSLTransformerErrorEvent.
getException() on page 10-33	Returns the transformation exception that XSLTransformer encountered an object unique id.
getMessage() on page 10-33	Returns the error message that describes the error encountered by XSLTransformer.

XSLTransformerErrorEvent()

Constructor for XSLTransformerErrorEvent.

Syntax

```
public  XSLTransformerErrorEvent( Object p0,
                                  Exception e);
```

Parameter	Description
p0	The Object that created this event.
e	The exception raised.

getException()

Returns the transformation exception that XSLTransformer encountered an object unique id.

Syntax

```
public Exception getException();
```

getMessage()

Returns the error message that describes the error encountered by XSLTransformer.

Syntax

```
public String getMessage();
```

XSLTransformerErrorListener Interface

This interface must be implemented in order to receive notifications about error events during the asynchronous transformation. The class implementing this interface must be added to the XSLTransformer using addXSLTransformerListener method.

Syntax

```
public interface XSLTransformerErrorListener extends
     java.util.EventListener
```

xslTransformerErrorCalled()

his method is called when parse or transformation error occurs.

Syntax

```
public void xslTransformerErrorCalled( XSLTransformerErrorEvent p0);
```

Parameter	Description
p0	The XSLTransformerErrorEvent object produced by the XSLTransformer.

XSLTransformerEvent Class

This class represents the event object used by XSLTransformer to notify XSL transformation events to all its registered listeners.

Syntax

```
public class XSLTransformerEvent extends java.util.EventObject
```

Table 10–15 Fields of XSLTransformerEvent

Field	Syntax	Description
id	protected int id	ID of the source XSLTransformer object

Table 10–16 Summary of Methods of XSLTransformerEvent

Method	Description
XSLTransformerEvent() on page 10-35	Constructs the XSLTransformerEvent object using the XSLTransformer source object and its unique id.
getID() on page 10-36	Returns unique id of the XSLTransformer object

XSLTransformerEvent()

Constructs the XSLTransformerEvent object using the XSLTransformer source object and its unique id.

Syntax

```
public  XSLTransformerEvent(java.lang.Object p0, int p1);
```

Parameter	Description
p0	The source XSLTransformer object that will fire the events
p1	Unique id identifying the source object

getID()

Returns unique id of the XSLTransformer object which can be used to identify which instance of the XSLTransformer generated this event in cases where multiple instances of XSLTransformer may be working in background.

Syntax

```
public int getID();
```

XSLTransformerListener Interface

Implemented to receive notifications of events during asynchronous transform. The implementing class must be added using addXSLTransformerListener method.

Syntax

```
public interface XSLTransformerListener extends java.util.EventListener
```

xslTransformerError()

This method is called when parse or transformation error occur.

Syntax

```
public void xslTransformerError( XSLTransformerEvent p0);
```

Parameter	Description
p0	The XSLTransformerEvent object produced by the XSLTransformer.

xslTransformerOver()

This method is called when the transformation is complete.

Syntax

```
public void xslTransformerOver( XSLTransformerEvent p0);
```

Parameter	Description
p0	The XSLTransformerEvent object produced by the XSLTransformer.

xslTransformerStarted()

This method is called when the transformation starts.

Syntax

```
public void xslTransformerStarted( XSLTransformerEvent p0);
```

Parameter	Description
p0	The XSLTransformerEvent object produced by the XSLTransformer.

oracle.xml.dbviewer Package

This is a visual bean used to display and edit HTML, XML or XSL file. XML files can be transferred between the file system, the database and the bean buffers. It can be used to transform database queries to XML. XML files can be parsed and transformed using XSL. The classes of the `oracle.xml.dbviewer` as summarized in Table 10–17.

Table 10–17 Summary of Classes of oracle.xml.dbviewer

Class	Description
DBViewer Class on page 10-40	Java bean that can be used to display database queries or any XML by applying XSL stylesheets and visualizing the resulting HTML in a scrollable swing panel.
DBViewerBeanInfo Class on page 10-62	This class provides information about the DBViewer Bean.

DBViewer Class

Java bean that can be used to display database queries or any XML by applying XSL stylesheets and visualizing the resulting HTML in a scrollable swing panel. This bean has tree buffers: XML, XSL and result buffer. The bean API enables the calling program to load/save the buffers from various sources and to apply stylesheet transformations to the XML buffer using the stylesheet in the XSL buffer. The result can be stored in the result buffer.

The XML and XSL buffer content can be shown as source or as a tree structures. The result buffer content can be rendered as HTML, and also shown as source or tree structures. The XML buffer can be loaded from database query.

All buffers can load and save files both from CLOB tables in Oracle database and from the file system. Therefore, the control can be used to move files between the file system and the user schema in the database.

Syntax

```
public class DBViewer extends javax.swing.JPanel implements java.io.Serializable
```

Table 10–18 Summary of Methods of DBViewer

Method	Description
DBViewer() on page 10-44	Constructs a new instance of DBViewer.
getHostname() on page 10-44	Retrieves the database host name.
getInstancename() on page 10-44	Retrieves the database instance name.
getPassword() on page 10-44	Retrieves the user password.
getPort() on page 10-44	Retrieves the database port number as a String.
getResBuffer() on page 10-44	Retrieves the content of the result buffer.
getResCLOBFileName() on page 10-44	Retrieves the result CLOB file name.
getResCLOBTableName() on page 10-45	Retrieves the result CLOB table name.
getResFileName() on page 10-45	Retrieves the result file name.
getUsername() on page 10-45	Retrieves the user name.

Table 10–18 (Cont.) Summary of Methods of DBViewer

Method	Description
getXmlBuffer() on page 10-45	Retrieves the content of the XML buffer.
getXmlCLOBFileName() on page 10-45	Retrieves the XML CLOB file name.
getXmlCLOBTableName() on page 10-45	Retrieves the XML CLOB table name.
getXmlFileName() on page 10-45	Retrieves the XML file name.
getXMLStringFromSQL() on page 10-46	Retrieves the XML presentation of a result set from SQL query as an XMLString.
getXslBuffer() on page 10-46	Retrieves the content of the XSL buffer.
getXslCLOBFileName() on page 10-46	Retrieves the XSL CLOB file name.
getXslCLOBTableName() on page 10-46	Retrieves XSL CLOB table name.
getXslFileName() on page 10-46	Retrieves XSL file name.
loadResBuffer() on page 10-46	Loads the result buffer.
loadResBufferFromClob() on page 10-47	Loads the result buffer from CLOB file.
loadResBufferFromFile() on page 10-47	Loads the result buffer from file.
loadXmlBuffer() on page 10-47	Loads the XML buffer.
loadXmlBufferFromClob() on page 10-48	Loads the XML buffer from CLOB file.
loadXmlBufferFromFile() on page 10-48	Loads the XML buffer from file.
loadXMLBufferFromSQL() on page 10-48	Loads the XML buffer from SQL result set.
loadXslBuffer() on page 10-49	Loads the XSL buffer from file.
loadXslBufferFromClob() on page 10-49	Loads the XSL buffer from CLOB file.
loadXslBufferFromFile() on page 10-49	Loads the XSL buffer from file.

Table 10–18 (Cont.) Summary of Methods of DBViewer

Method	Description
parseResBuffer() on page 10-49	Parses the result buffer, refreshes the tree and source views, and returns the XMLDocument.
parseXmlBuffer() on page 10-50	Parse the XML buffer, refreshes the tree and source views, and returns the XMLDocument.
parseXslBuffer() on page 10-50	Parses the XSL buffer, refreshes the tree and source views, and returns the XMLDocument.
saveResBuffer() on page 10-50	Saves the result buffer to file.
saveResBufferToClob() on page 10-50	Saves the result buffer to CLOB file.
saveResBufferToFile() on page 10-51	Saves the result buffer to file.
saveXmlBuffer() on page 10-51	Saves the XML buffer to file.
saveXmlBufferToClob() on page 10-51	Saves the XML buffer to CLOB file.
saveXmlBufferToFile() on page 10-51	Saves the XML buffer to file.
saveXslBuffer() on page 10-52	Saves the XSL buffer to file.
saveXslBufferToClob() on page 10-52	Saves the XSL buffer to CLOB file.
saveXslBufferToFile() on page 10-52	Saves the XSL buffer to file.
setHostname() on page 10-52	Sets database host name.
setInstancename() on page 10-53	Sets database instance name.
setPassword() on page 10-53	Sets user password.
setPort() on page 10-53	Sets database port number.
setResBuffer() on page 10-54	Sets new text in the result buffer.
setResCLOBFileName() on page 10-54	Sets result CLOB file name.
setResCLOBTableName() on page 10-54	Sets result CLOB table name.
setResFileName() on page 10-54	Sets result file name.
setResHtmlView() on page 10-55	Show the result buffer as rendered HTML.

Table 10–18 (Cont.) Summary of Methods of DBViewer

Method	Description
setResSourceEditView() on page 10-55	Shows the result buffer as XML source and enter edit mode.
setResSourceView() on page 10-55	Shows the result buffer as XML source.
setResTreeView() on page 10-56	Shows the result buffer as an XML tree view.
setUsername() on page 10-56	Sets the user name.
setXmlBuffer() on page 10-56	Sets new text in the XML buffer.
setXmlCLOBFileName() on page 10-57	Sets XML CLOB file name.
setXmlCLOBTableName() on page 10-57	Sets the XML CLOB table name.
setXmlFileName() on page 10-57	Sets the XML file name.
setXmlSourceEditView() on page 10-57	Shows the XML buffer as XML source and enter edit mode.
setXmlSourceView() on page 10-58	Shows the XML buffer as an XML source.
setXmlTreeView() on page 10-58	Shows the XML buffer as tree.
setXslBuffer() on page 10-58	Sets new text in the XSL buffer.
setXslCLOBFileName() on page 10-59	Sets the XSL CLOB file name.
setXslCLOBTableName() on page 10-59	Sets the XSL CLOB table name.
setXslFileName() on page 10-59	Sets the XSL file name.
setXslSourceEditView() on page 10-60	Shows the XSL buffer as XML source and enter edit mode.
setXslSourceView() on page 10-60	Shows the XSL buffer as an XML source.
setXslTreeView() on page 10-60	Shows the XSL buffer as tree.
transformToDoc() on page 10-60	Transforms the content of the XML buffer by applying the stylesheet from the XSL buffer.
transformToRes() on page 10-61	Applies the stylesheet transformation from the XSL buffer to the XML in the XML buffer, and stores the result in the result buffer.
transformToString() on page 10-61	Transforms the content of the XML buffer by applying the stylesheet from the XSL buffer.

DBViewer()

Constructs a new instance of DBViewer.

Syntax

```
public  DBViewer();
```

getHostname()

Retrieves the database host name.

Syntax

```
public java.lang.String getHostname();
```

getInstancename()

Retrieves the database instance name.

Syntax

```
public java.lang.String getInstancename();
```

getPassword()

Retrieves the user password.

Syntax

```
public java.lang.String getPassword();
```

getPort()

Retrieves the database port number as a String.

Syntax

```
public java.lang.String getPort();
```

getResBuffer()

Retrieves the content of the result buffer.

Syntax

```
public java.lang.String getResBuffer();
```

getResCLOBFileName()

Retrieves the result CLOB file name.

Syntax

```
public java.lang.String getResCLOBFileName();
```

getResCLOBTableName()

Retrieves the result CLOB table name.

Syntax

```
public java.lang.String getResCLOBTableName();
```

getResFileName()

Retrieves the result file name.

Syntax

```
public java.lang.String getResFileName();
```

getUsername()

Retrieves the user name.

Syntax

```
public java.lang.String getUsername();
```

getXmlBuffer()

Retrieves the content of the XML buffer.

Syntax

```
public java.lang.String getXmlBuffer();
```

getXmlCLOBFileName()

Retrieves the XML CLOB file name.

Syntax

```
public java.lang.String getXmlCLOBFileName();
```

getXmlCLOBTableName()

Retrieves the XML CLOB table name.

Syntax

```
public java.lang.String getXmlCLOBTableName();
```

getXmlFileName()

Retrieves the XML file name.

Syntax

```
public java.lang.String getXmlFileName();
```

getXMLStringFromSQL()

Retrieves the XML presentation of a result set from SQL query as an XMLString.

Syntax

```
public java.lang.String getXMLStringFromSQL( java.lang.String sqlText);
```

Parameter	Description
sqlText	The SQL text.

getXslBuffer()

Retrieves the content of the XSL buffer.

Syntax

```
public java.lang.String getXslBuffer();
```

getXslCLOBFileName()

Retrieves the XSL CLOB file name.

Syntax

```
public java.lang.String getXslCLOBFileName();
```

getXslCLOBTableName()

Retrieves XSL CLOB table name.

Syntax

```
public java.lang.String getXslCLOBTableName();
```

getXslFileName()

Retrieves XSL file name.

Syntax

```
public java.lang.String getXslFileName();
```

loadResBuffer()

Loads the result buffer. The options are described in the following table.

Syntax	Description
`public void loadResBuffer( String filename);`	Loads the result buffer from file.
`public void loadResBuffer( String clobTablename, String clobFilename);`	Loads the result buffer from a CLOB file.
`public void loadResBuffer( oracle.xml.parser.v2.XMLDocument resDoc);`	Loads the result buffer from an XMLDocument.

Parameter	Description
`filename`	The file name.
`clobTablename`	The CLOB table name.
`clobFilename`	The CLOB file name.
`resDoc`	The XMLDocument.

loadResBufferFromClob()

Loads the result buffer from CLOB file.

Syntax

```
public void loadResBufferFromClob();
```

loadResBufferFromFile()

Loads the result buffer from file.

Syntax

```
public void loadResBufferFromFile();
```

loadXmlBuffer()

Loads the XML buffer. The options are described in the following table.

Syntax	Description
`public void loadXmlBuffer( j String filename);`	Loads the XML buffer from file.

Syntax	Description
`public void loadXmlBuffer( String clobTablename, String clobFilename);`	Loads the XML buffer from CLOB file.
`public void loadXmlBuffer( oracle.xml.parser.v2.XMLDocument xmlDoc);`	Loads the XML buffer from XMLDocument.

Parameter	Description
`filename`	The file name.
`clobTablename`	The CLOB table name.
`clobFilename`	The CLOB file name.
`resDoc`	The XMLDocument.

loadXmlBufferFromClob()

Loads the XML buffer from CLOB file.

Syntax

```
public void loadXmlBufferFromClob();
```

loadXmlBufferFromFile()

Loads the XML buffer from file.

Syntax

```
public void loadXmlBufferFromFile();
```

loadXMLBufferFromSQL()

Loads the XML buffer from SQL result set.

Syntax

```
public void loadXMLBufferFromSQL( String sqlText);
```

Parameter	Description
`sqlText`	SQL text

loadXslBuffer()

Loads the XSL buffer from file. The options are described in the following table.

Syntax	Description
`public void loadXslBuffer( String filename);`	Loads the XSL buffer from file.
`public void loadXslBuffer( String clobTablename, String clobFilename);`	Loads the XSL buffer from CLOB file.
`public void loadXslBuffer( oracle.xml.parser.v2.XMLDocument xslDoc);`	Loads the XSL buffer from XMLDocument.

Parameter	Description
`filename`	The file name.
`clobTablename`	The CLOB table name.
`clobFilename`	The CLOB file name.
`xslDoc`	The XMLDocument.

loadXslBufferFromClob()

Loads the XSL buffer from CLOB file.

Syntax

```
public void loadXslBufferFromClob();
```

loadXslBufferFromFile()

Loads the XSL buffer from file.

Syntax

```
public void loadXslBufferFromFile();
```

parseResBuffer()

Parses the result buffer, refreshes the tree and source views, and returns the XMLDocument.

Syntax

```
public oracle.xml.parser.v2.XMLDocument parseResBuffer();
```

parseXmlBuffer()

Parse the XML buffer, refreshes the tree and source views, and returns the XMLDocument.

Syntax

```
public oracle.xml.parser.v2.XMLDocument parseXmlBuffer();
```

parseXslBuffer()

Parses the XSL buffer, refreshes the tree and source views, and returns the XMLDocument.

Syntax

```
public oracle.xml.parser.v2.XMLDocument parseXslBuffer();
```

saveResBuffer()

Saves the result buffer to file. The options are described in the following table.

Syntax	Description
`public void saveResBuffer( String filename);`	Save the result buffer to file.
`public void saveResBuffer( String tablename, String filename);`	Save the result buffer to CLOB file.

Parameter	Description
`tablename`	The CLOB table name.
`filename`	The CLOB file name.

saveResBufferToClob()

Saves the result buffer to CLOB file.

Syntax

```
public void saveResBufferToClob();
```

saveResBufferToFile()

Saves the result buffer to file.

Syntax

```
public void saveResBufferToFile();
```

saveXmlBuffer()

Saves the XML buffer to file. The options are described in the following table.

Syntax	Description
`public void saveXmlBuffer( String filename);`	Saves the XML buffer to file.
`public void saveXmlBuffer( String tablename, String filename);`	Saves the XML buffer to CLOB file.

Parameter	Description
`tablename`	The CLOB table name.
`filename`	The CLOB file name.

saveXmlBufferToClob()

Saves the XML buffer to CLOB file.

Syntax

```
public void saveXmlBufferToClob();
```

saveXmlBufferToFile()

Saves the XML buffer to file.

Syntax

```
public void saveXmlBufferToFile();
```

saveXslBuffer()

Saves the XSL buffer to file. The options are described in the following table.

Syntax	Description
`public void saveXslBuffer( String fileName);`	Saves the XSL buffer to file in the file system.
`public void saveXslBuffer( String tableName, String fileName);`	Saves the XSL buffer to a CLOB file.

Parameter	Description
`tableName`	The table name.
`fileName`	The file name.

saveXslBufferToClob()

Saves the XSL buffer to CLOB file.

Syntax

```
public void saveXslBufferToClob();
```

saveXslBufferToFile()

Saves the XSL buffer to file.

Syntax

```
public void saveXslBufferToFile();
```

setHostname()

Sets database host name.

Syntax

```
public void setHostname( java.lang.String hostname);
```

Parameter	Description
hostname	The host name.

setInstancename()

Sets database instance name.

Syntax

```
public void setInstancename( String instancename);
```

Parameter	Description
instancename	The database instance name.

setPassword()

Sets user password.

Syntax

```
public void setPassword( String password);
```

Parameter	Description
password	The user password.

setPort()

Sets database port number.

Syntax

```
public void setPort( String port);
```

Parameter	Description
port	String containing the port number.

setResBuffer()

Sets new text in the result buffer.

Syntax

```
public void setResBuffer( String text);
```

Parameter	Description
text	The new text.

setResCLOBFileName()

Sets result CLOB file name.

Syntax

```
public void setResCLOBFileName( String name);
```

Parameter	Description
name	Result CLOB file name.

setResCLOBTableName()

Sets result CLOB table name.

Syntax

```
public void setResCLOBTableName( String name);
```

Parameter	Description
name	Result CLOB table name.

setResFileName()

Sets result file name.

Syntax

```
public void setResFileName( String name);
```

Parameter	Description
name	Result file name.

setResHtmlView()

Show the result buffer as rendered HTML.

Syntax

```
public void setResHtmlView( boolean on);
```

Parameter	Description
on	Switch to show result buffer a HTML; TRUE to show, FALSE otherwise.

setResSourceEditView()

Shows the result buffer as XML source and enter edit mode.

Syntax

```
public void setResSourceEditView( boolean on);
```

Parameter	Description
on	Switch to show result buffer as HTML with edit mode; TRUE to show.

setResSourceView()

Shows the result buffer as XML source.

Syntax

```
public void setResSourceView( boolean on);
```

Parameter	Description
on	Switch to show result buffer as XML source; TRUE to show, FALSE otherwise.

setResTreeView()

Shows the result buffer as an XML tree view.

Syntax

```
public void setResTreeView( boolean on);
```

Parameter	Description
on	Switch to show result buffer as XML tree; TRUE to show, FALSE otherwise.

setUsername()

Sets the user name.

Syntax

```
public void setUsername( String username);
```

Parameter	Description
username	The user name.

setXmlBuffer()

Sets new text in the XML buffer.

Syntax

```
public void setXmlBuffer( String text)
```

Parameter	Description
text	The XML text

setXmlCLOBFileName()

Sets XML CLOB file name.

Syntax

```
public void setXmlCLOBFileName( String name);
```

Parameter	Description
name	The XML CLOB file name.

setXmlCLOBTableName()

Sets the XML CLOB table name.

Syntax

```
public void setXmlCLOBTableName( String name);
```

Parameter	Description
name	The XML CLOB table name.

setXmlFileName()

Sets the XML file name.

Syntax

```
public void setXmlFileName( String name);
```

Parameter	Description
name	The XML file name.

setXmlSourceEditView()

Shows the XML buffer as XML source and enter edit mode.

Syntax

```
public void setXmlSourceEditView( boolean on);
```

Parameter	Description
on	Switch to show XML buffer as XML source with edit mode; TRUE to show.

setXmlSourceView()

Shows the XML buffer as an XML source.

Syntax

```
public void setXmlSourceView( boolean on);
```

Parameter	Description
on	Switch to show XML buffer as XML source; TRUE to show, FALSE otherwise.

setXmlTreeView()

Shows the XML buffer as tree.

Syntax

```
public void setXmlTreeView( boolean on);
```

Parameter	Description
on	Switch to show XML buffer as XML tree; TRUE to show, FALSE otherwise.

setXslBuffer()

Sets new text in the XSL buffer.

Syntax

```
public void setXslBuffer( String text);
```

Parameter	Description
text	The XSL text.

setXslCLOBFileName()

Sets the XSL CLOB file name.

Syntax

```
public void setXslCLOBFileName( String name);
```

Parameter	Description
name	The XSL CLOB file name.

setXslCLOBTableName()

Sets the XSL CLOB table name.

Syntax

```
public void setXslCLOBTableName( String name);
```

Parameter	Description
name	The XSL CLOB table name.

setXslFileName()

Sets the XSL file name.

Syntax

```
public void setXslFileName( String name);
```

Parameter	Description
name	The XSL file name.

setXslSourceEditView()

Shows the XSL buffer as XML source and enter edit mode.

Syntax

```
public void setXslSourceEditView( boolean on);
```

Parameter	Description
on	Switch to show XSL buffer as XML source with edit mode; TRUE to show.

setXslSourceView()

Shows the XSL buffer as an XML source.

Syntax

```
public void setXslSourceView( boolean on);
```

Parameter	Description
on	Switch to show XSL buffer as XML source; TRUE to show, FALSE otherwise.

setXslTreeView()

Shows the XSL buffer as tree.

Syntax

```
public void setXslTreeView( boolean on);
```

Parameter	Description
on	Switch to show the XSL buffer as XML tree; TRUE to show, FALSE otherwise.

transformToDoc()

Transforms the content of the XML buffer by applying the stylesheet from the XSL buffer.

Syntax

```
public oracle.xml.parser.v2.XMLDocument transformToDoc();
```

transformToRes()

Applies the stylesheet transformation from the XSL buffer to the XML in the XML buffer, and stores the result in the result buffer.

Syntax

```
public void transformToRes();
```

transformToString()

Transfroms the content of the XML buffer by applying the stylesheet from the XSL buffer.

Syntax

```
public java.lang.String transformToString();
```

DBViewerBeanInfo Class

This class provides information about the DBViewer Bean.

Syntax

```
public class DBViewerBeanInfo extends java.beans.SimpleBeanInfo

java.lang.Object
```

Table 10–19 Summary of Methods of DBViewerBeanInfo

Method	Description
DBViewerBeanInfo() on page 10-62	Class constructor.
getIcon() on page 10-62	Retrieves an image of the requested icon type.
getPropertyDescriptors() on page 10-62	Retrieves an array of DBViwer bean's editable PropertyDescriptors.

DBViewerBeanInfo()

Class constructor.

Syntax

```
public  DBViewerBeanInfo();
```

getIcon()

Retrieves an image object representing the requested icon type for DBViewer bean in toolbars, toolboxes, and so on.

Syntax

```
public java.awt.Image getIcon( int iconKind);
```

Parameter	Description
iconKind	The kind of icon requested.

getPropertyDescriptors()

Retrieves an array of DBViwer bean's editable PropertyDescriptors.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

oracle.xml.srcviewer Package

This is a visual bean used to display an XML source document with syntax highlighting. The color, fonts and sizes of different XML language element are customizable.

The classes of the `oracle.xml.srcviewer` as summarized in Table 10–20.

Table 10–20 Summary of Classes of oracle.xml.srcviewer

Class	Description
XMLSourceView Class on page 10-65	Shows an XML document.
XMLSourceViewBeanInfo Class on page 10-81	This class provides information about the XMLSourceView Bean.

XMLSourceView Class

Shows an XML document. Recognizes the following XML token types: Tag, Attribute Name, Attribute Value, Comment, CDATA, PCDATA, PI Data, PI Name and NOTATION Symbol. Each token type has a foreground color and font. The default color/font settings can be changed by the user. Takes as input an org.w3c.dom.Document object.

Syntax

```
public class XMLSourceView extends javax.swing.JPanel implements
java.io.Serializable
```

Table 10–21 Fields of ElementDecl

Field	Syntax	Description
`inputDOMDocument`	`protected org.w3c.dom.Document inputDOMDocument`	XML Document to be displayed.
`jScrollPane`	`protected javax.swing.JScrollPane jScrollPane`	The java swing component used by XMLSourveView to enable scrolling.
`jTextPane`	`protected javax.swing.JTextPane jTextPane`	The java swing component used by XMLSourceView to display text.
`xmlStyledDocument`	`protected oracle.xml.srcviewer.XMLStyledDocument xmlStyledDocument`	Represents a stylized XML document; associates XML tokens with attributes, such as font and color.

Table 10–22 Summary of Methods of XMLSourceView

Method	Description
XMLSourceView() on page 10-68	The class constructor. Creates an object of type XMLSourceView.

Table 10–22 (Cont.) Summary of Methods of XMLSourceView

Method	Description
fontGet() on page 10-68	Extracts and returns the font from a given attribute set.
fontSet() on page 10-68	Sets the mutable attribute set font.
getAttributeNameFont() on page 10-69	Returns the Attribute Value font.
getAttributeNameForeground() on page 10-69	Returns the Attribute Name foreground color.
getAttributeValueFont() on page 10-69	Returns the Attribute Value font.
getAttributeValueForeground() on page 10-69	Returns the Attribute Value foreground color.
getBackground() on page 10-69	Returns the background color.
getCDATAFont() on page 10-70	Returns the CDATA font.
getCDATAForeground() on page 10-70	Returns the CDATA foreground color.
;getCommentDataFont() on page 10-70	Returns the Comment Data font.
getCommentDataForeground() on page 10-70	Returns the Comment Data foreground color.
getEditedText() on page 10-70	Returns the edited text.
getJTextPane() on page 10-70	Returns the viewer `JTextPane` component used by XMLSourceViewer.
getMinimumSize() on page 10-71	Returns the XMLSourceView minimal size.
getNodeAtOffset() on page 10-71	Returns the XML node at a given offset.
getPCDATAFont() on page 10-71	Returns the PCDATA font.
getPCDATAForeground() on page 10-71	Returns the PCDATA foreground color.
getPIDataFont() on page 10-71	Returns the PI Data font.
getPIDataForeground() on page 10-71	Returns the PI Data foreground color.
getPINameFont() on page 10-72	Returns the PI Name font.
getPINameForeground() on page 10-72	Returns the PI Data foreground color.

Table 10–22 (Cont.) Summary of Methods of XMLSourceView

Method	Description
getSymbolFont() on page 10-72	Returns the NOTATION Symbol font.
getSymbolForeground() on page 10-72	Returns the NOTATION Symbol foreground color.
getTagFont() on page 10-72	Returns the Tag font.
getTagForeground() on page 10-72	Returns the Tag foreground color.
getText() on page 10-72	Returns the XML document as a String.
isEditable() on page 10-73	Returns boolean to indicate whether this object is editable.
selectNodeAt() on page 10-73	Moves the cursor to XML Node at specified offset.
setAttributeNameFont() on page 10-73	Sets the Attribute Name font.
setAttributeNameForeground() on page 10-73	Sets the Attribute Name foreground color.
setAttributeValueFont() on page 10-74	Sets the Attribute Value font.
setAttributeValueForeground() on page 10-74	Sets the Attribute Value foreground color.
setBackground() on page 10-74	Sets the background color.
setCDATAFont() on page 10-75	Sets the CDATA font.
setCDATAForeground() on page 10-75	Sets the CDATA foreground color.
setCommentDataFont() on page 10-75	Sets the Comment font.
setCommentDataForeground() on page 10-75	Sets the Comment foreground color.
setEditable() on page 10-76	Sets the specified boolean to indicate whether this object should be editable.
setPCDATAFont() on page 10-76	Sets the PCDATA font.
setPCDATAForeground() on page 10-76	Sets the PCDATA foreground color.
setPIDataFont() on page 10-77	Sets the PI Data font.
setPIDataForeground() on page 10-77	Sets the PI Data foreground color.

Table 10–22 (Cont.) Summary of Methods of XMLSourceView

Method	Description
setPINameFont() on page 10-77	Sets the PI Name font.
setPINameForeground() on page 10-78	Sets the PI Name foreground color.
setSelectedNode() on page 10-78	Sets the cursor position at the selected XML node.
setSymbolFont() on page 10-78	Sets the NOTATION Symbol font.
setSymbolForeground() on page 10-78	Sets the NOTATION Symbol foreground color.
setTagFont() on page 10-79	Sets the Tag font.
setTagForeground() on page 10-79	Sets the Tag foreground color.
setXMLDocument() on page 10-79	Associates the XMLviewer with a XML document.

XMLSourceView()

The class constructor. Creates an object of type XMLSourceView.

Syntax

```
public  XMLSourceView();
```

fontGet()

Extracts and returns the font from a given attribute set.

Syntax

```
public static java.awt.Font fontGet(
                javax.swing.text.AttributeSet attributeSet);
```

Parameter	Description
attributeSet	The source AttributeSet.

fontSet()

Sets the mutable attribute set font.

Syntax

```
public static void fontSet(
                javax.swing.text.MutableAttributeSet mutAttributeSet,
                java.awt.Font font);
```

Parameter	Description
mutAttributeSet	The mutableattributeset to update.
font	The new Font for the mutableattributeset.

getAttributeNameFont()

Returns the Attribute Value font.

Syntax

```
public java.awt.Font getAttributeNameFont();
```

getAttributeNameForeground()

Returns the Attribute Name foreground color.

Syntax

```
public java.awt.Color getAttributeNameForeground();
```

getAttributeValueFont()

Returns the Attribute Value font.

Syntax

```
public java.awt.Font getAttributeValueFont();
```

getAttributeValueForeground()

Returns the Attribute Value foreground color.

Syntax

```
public java.awt.Color getAttributeValueForeground();
```

getBackground()

Returns the background color. Overrides getBackground() in java.awt.Component class.

Syntax

```
public java.awt.Color getBackground();
```

getCDATAFont()

Returns the CDATA font.

Syntax

```
public java.awt.Font getCDATAFont();
```

getCDATAForeground()

Returns the CDATA foreground color.

Syntax

```
public java.awt.Color getCDATAForeground();
```

;getCommentDataFont()

Returns the Comment Data font.

Syntax

```
public java.awt.Font getCommentDataFont();
```

getCommentDataForeground()

Returns the Comment Data foreground color.

Syntax

```
public java.awt.Color getCommentDataForeground();
```

getEditedText()

Returns the edited text.

Syntax

```
public java.lang.String getEditedText();
```

getJTextPane()

Returns the viewer `JTextPane` component used by XMLSourceViewer.

Syntax

```
public javax.swing.JTextPane getJTextPane();
```

getMinimumSize()

Returns the XMLSourceView minimal size. Overrides getMinimumSize() in javax.swing.JComponent class.

Syntax

```
public java.awt.Dimension getMinimumSize();
```

getNodeAtOffset()

Returns the XML node at a given offset.

Syntax

```
public org.w3c.dom.Node getNodeAtOffset( int i);
```

Parameter	Description
i	The node offset.

getPCDATAFont()

Returns the PCDATA font.

Syntax

```
public java.awt.Font getPCDATAFont();
```

getPCDATAForeground()

Returns the PCDATA foreground color.

Syntax

```
public java.awt.Color getPCDATAForeground();
```

getPIDataFont()

Returns the PI Data font.

Syntax

```
public java.awt.Font getPIDataFont();
```

getPIDataForeground()

Returns the PI Data foreground color.

Syntax

```
public java.awt.Color getPIDataForeground();
```

getPINameFont()

Returns the PI Name font.

Syntax

```
public java.awt.Font getPINameFont();
```

getPINameForeground()

Returns the PI Data foreground color.

Syntax

```
public java.awt.Color getPINameForeground();
```

getSymbolFont()

Returns the NOTATION Symbol font.

Syntax

```
public java.awt.Font getSymbolFont();
```

getSymbolForeground()

Returns the NOTATION Symbol foreground color.

Syntax

```
public java.awt.Color getSymbolForeground();
```

getTagFont()

Returns the Tag font.

Syntax

```
public java.awt.Font getTagFont();
```

getTagForeground()

Returns the Tag foreground color.

Syntax

```
public java.awt.Color getTagForeground();
```

getText()

Returns the XML document as a String.

Syntax

```
public java.lang.String getText();
```

isEditable()

Returns boolean to indicate whether this object is editable.

Syntax

```
public boolean isEditable();
```

selectNodeAt()

Moves the cursor to XML Node at the specified offset.

Syntax

```
public void selectNodeAt( int i);
```

Parameter	Description
i	The node offset.

setAttributeNameFont()

Sets the Attribute Name font.

Syntax

```
public void setAttributeNameFont( java.awt.Font font);
```

Parameter	Description
font	The new Font Attribute Name.

setAttributeNameForeground()

Sets the Attribute Name foreground color.

Syntax

```
public void setAttributeNameForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for Attribute Name.

setAttributeValueFont()

Sets the Attribute Value font.

Syntax

```
public void setAttributeValueFont( java.awt.Font font);
```

Parameter	Description
font	The new Font Attribute Value

setAttributeValueForeground()

Sets the Attribute Value foreground color.

Syntax

```
public void setAttributeValueForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for Attribute Value.

setBackground()

Sets the background color. Overrides `setBackground()` in `javax.swing.JComponent` class.

Syntax

```
public void setBackground( java.awt.Color color);
```

Parameter	Description
font	The new background Color.

setCDATAFont()

Sets the CDATA font.

Syntax

```
public void setCDATAFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for CDATA.

setCDATAForeground()

Sets the CDATA foreground color.

Syntax

```
public void setCDATAForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for CDATA.

setCommentDataFont()

Sets the Comment font.

Syntax

```
public void setCommentDataFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for the XML Comments.

setCommentDataForeground()

Sets the Comment foreground color.

Syntax

```
public void setCommentDataForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for Comment.

setEditable()

Sets the specified boolean to indicate whether this object should be editable.

Syntax

```
public void setEditable( boolean edit);
```

Parameter	Description
edit	Flag indicating if the object should be editable; TRUE when text displayed can be edited, FALSE otherwise.

setPCDATAFont()

Sets the PCDATA font.

Syntax

```
public void setPCDATAFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for PCDATA.

setPCDATAForeground()

Sets the PCDATA foreground color.

Syntax

```
public void setPCDATAForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for PCDATA.

setPIDataFont()

Sets the PI Data font.

Syntax

```
public void setPIDataFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for PI Data.

setPIDataForeground()

Sets the PI Data foreground color.

Syntax

```
public void setPIDataForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for PI Data.

setPINameFont()

Sets the PI Name font.

Syntax

```
public void setPINameFont(java.awt.Font font);
```

Parameter	Description
font	The new Font for the PI Names.

setPINameForeground()

Sets the PI Name foreground color.

Syntax

```
public void setPINameForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for PI Name.

setSelectedNode()

Sets the cursor position at the selected XML node.

Syntax

```
public void setSelectedNode( org.w3c.dom.Node node);
```

Parameter	Description
node	The selected node.

setSymbolFont()

Sets the NOTATION Symbol font.

Syntax

```
public void setSymbolFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for NOTATION Symbol.

setSymbolForeground()

Sets the NOTATION Symbol foreground color.

Syntax

```
public void setSymbolForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for NOTATION Symbol.

setTagFont()

Sets the Tag font.

Syntax

```
public void setTagFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for the XML Tags.

setTagForeground()

Sets the Tag foreground color.

Syntax

```
public void setTagForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for the XML Tags.

setXMLDocument()

Associates the XMLviewer with a XML document.

Syntax

```
public void setXMLDocument( org.w3c.dom.Document document);
```

Parameter	Description
document	The Document to display.

XMLSourceViewBeanInfo Class

This class provides information about the XMLSourceView Bean.

Syntax

```
public class XMLSourceViewBeanInfo extends java.beans.SimpleBeanInfo
```

Table 10–23 Summary of Methods of XMLSourceViewBeanInfo

Method	Description
XMLSourceViewBeanInfo() on page 10-81	Class constructor.
getIcon() on page 10-81	Retrieves an image object representing the requested icon type for XMLSourceView bean in toolbars, toolboxes, and so on.
getPropertyDescriptors() on page 10-82	Retrieves an array of XMLSourveView bean's editable PropertyDescriptors.

XMLSourceViewBeanInfo()

Class constructor.

Syntax

```
public  XMLSourceViewBeanInfo();
```

getIcon()

Retrieves an image object representing the requested icon type for XMLSourceView bean in toolbars, toolboxes, and so on. Overrides `getIcon()` in `java.beans.SimpleBeanInfo` class.

Syntax

```
public java.awt.Image getIcon( int iconKind);
```

Parameter	Description
`iconKind`	The kind of icon requested.

getPropertyDescriptors()

Retrieves an array of XMLSourveView bean's editable PropertyDescriptors. Overrides `getPropertyDescriptors()` in `java.beans.SimpleBeanInfo` Class.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

oracle.xml.transviewer Package

This is a visual bean. It allows users to load XML and XSL buffers from the file system or from the database's CLOB tables. The XML buffer can be transformed using the XSL buffer. XML, XSL or HTML buffers can be saved in the file system or in the database as CLOB tables. Each CLOB table has two columns, string type to hold the filename and CLOB type to hold the file data. CLOB tables can be created or deleted. The XML and XSL buffers can be edited and parsed.

The classes of the `oracle.xml.srcviewer` as summarized in Table 10–24.

Table 10–24 Summary of Classes of oracle.xml.transviewer

Class	Description
DBAccess Class on page 10-84	Maintains CLOB tables that can hold multiple XML and text documents.
DBAccessBeanInfo Class on page 10-92	This class provides information about the DBAccess Bean.
XMLTransformPanel Class on page 10-94	Applies XSL transformations to XML documents and visualizes the result.
XMLTransformPanelBeanInfo Class on page 10-95	This class provides information about the XMLTransformPanel Bean
XMLTransViewer Class on page 10-97	Simple application that uses XMLTransformPanel. Can be used from the command line to edit and parse XML files, edit and apply XSL transformations and retrieve and save XML, XSL and result files in the file system or in the database.

DBAccess Class

Maintains CLOB tables that can hold multiple XML and text documents. Each table is created using the statement:

```
CREATE TABLE tablename FILENAME CHAR( 16) UNIQUE, FILEDATA CLOB) LOB(FILEDATA)
STORE AS (DISABLE STORAGE IN ROW).
```

- Each XML (or text) document is stored as a row in the table and the `FILENAME` field holds a unique string that is used as a key to retrieve, update or delete the row.
- The document text is stored in the `FILEDATA` field that is a CLOB object.
- These CLOB tables are automatically maintained by the transviewer bean.
- The CLOB tables maintained by this class can be used by the transviewer bean.
- The class creates and deletes CLOB tables, lists a CLOB table content, and also adds, replaces or deletes text documents in these CLOB tables.

Syntax

```
public class DBAccess extends java.lang.Object
```

Table 10–25 Summary of Methods of DBAcceess

Method	Description
DBAccess() on page 10-85	Class constructor.
createBLOBTable() on page 10-85	Create BLOB table, returning `TRUE` if successful.
createXMLTable() on page 10-86	Creates XML table, returning `TRUE` if successful.
deleteBLOBName() on page 10-86	Deletes binary file from BLOB table, returning `TRUE` if successful.
deleteXMLName() on page 10-86	Delete file from XML table, returning `TRUE` if successful.
dropBLOBTable() on page 10-87	Deletes BLOB table. `TRUE` if successful.

Table 10–25 (Cont.) Summary of Methods of DBAcceess

Method	Description
dropXMLTable() on page 10-87	Deletes XML table. Returns `TRUE` if successful.
getBLOBData() on page 10-88	Retrieves binary file from BLOB table as a byte array. Returns `TRUE` if successful.
getNameSize() on page 10-88	Returns the size of the field where the filename is kept.
getXMLData() on page 10-88	Retrieves text file from XML table as a String.
getXMLNames() on page 10-89	Returns all file names in XML table as a String array.
getXMLTableNames() on page 10-89	Retrieves an array of all XML table names, starting with a user-specified string.
insertBLOBData() on page 10-89	Inserts binary file as a row in BLOB table. Returns `TRUE` if successful.
insertXMLData() on page 10-90	Inserts text file as a row in XML table. Returns `TRUE` if successful.
isXMLTable() on page 10-90	Checks if the table is an XML table, in which case returns `TRUE`.
replaceXMLData() on page 10-91	Replaces text file as a row in XML table, returning `TRUE` if successful.
xmlTableExists() on page 10-91	Checks if the XML table exists, returning `TRUE` if successful.

DBAccess()

Class constructor.

Syntax

```
public  DBAccess();
```

createBLOBTable()

Create BLOB table, returning `TRUE` if successful.

Syntax

```
public boolean createBLOBTable( Connection con,
                                String tableName);
```

Parameter	Description
con	The Connection object.
tableName	The table name.

createXMLTable()

Creates XML table, returning TRUE if successful.

Syntax

```
public boolean createXMLTable( Connection con,
                               String tableName);
```

Parameter	Description
con	The Connection object.
tableName	The table name.

deleteBLOBName()

Deletes binary file from BLOB table, returning TRUE if successful.

Syntax

```
public boolean deleteBLOBName( java.sql.Connection con,
                              String tableName,
                               String blobName);
```

Parameter	Description
con	The Connection object.
tableName	The table name.
blobName	The file name.

deleteXMLName()

Delete file from XML table, returning TRUE if successful.

Syntax

```
public boolean deleteXMLName( java.sql.Connection con,
                              String tableName,
                              String xmlName);
```

Parameter	Description
con	The Connection object.
tableName	The table name.
xmlName	The file name.

dropBLOBTable()

Deletes BLOB table. TRUE if successful.

Syntax

```
public boolean dropBLOBTable( java.sql.Connection con,
                              String tableName);
```

Parameter	Description
con	The Connection object.
tableName	The table name.

dropXMLTable()

Deletes XML table. Returns TRUE if successful.

Syntax

```
public boolean dropXMLTable( java.sql.Connection con,
                 java.lang.String tableName);
```

Parameter	Description
con	The Connection object.
tableName	The table name.

getBLOBData()

Retrieves binary file from BLOB table as a byte array. Returns `TRUE` if successful.

Syntax

```
public byte[] getBLOBData( java.sql.Connection con,
                           String tableName,
                           String xmlName);
```

Parameter	Description
`con`	The Connection object.
`tableName`	The table name.
`xmlName`	The file name.

getNameSize()

Returns the size of the field where the filename is kept.

Syntax

```
public int getNameSize();
```

getXMLData()

Retrieves text file from XML table as a String.

Syntax

```
public java.lang.String getXMLData( java.sql.Connection con,
                                    String tableName,
                                    String xmlName);
```

Parameter	Description
`con`	The Connection object.
`tableName`	The table name.
`xmlName`	The file name.

getXMLNames()

Returns all file names in XML table as a String array.

Syntax

```
public java.lang.String[] getXMLNames( java.sql.Connection con,
                                       String tableName);
```

Parameter	Description
con	The Connection object.
tableName	The table name.

getXMLTableNames()

Retrieves an array of all XML table names, starting with a user-specified string.

Syntax

```
public java.lang.String[] getXMLTableNames( java.sql.Connection con,
                                            String tablePrefix);
```

Parameter	Description
con	The Connection object.
tablePrefix	The table prefix string that starts the retrieved array of XML table names.

insertBLOBData()

Inserts binary file as a row in BLOB table. Returns TRUE if successful.

Syntax

```
public boolean insertBLOBData( java.sql.Connection con,
                               String tableName,
                               String xmlName,
                               byte[] xmlData);
```

Parameter	Description
con	The Connection object.
tableName	The table name.
xmlName	The file name.
xmlData	The byte array with file data.

insertXMLData()

Inserts text file as a row in XML table. Returns TRUE if successful.

Syntax

```
public boolean insertXMLData( java.sql.Connection con,
                              String tableName,
                              String xmlName,
                              String xmlData);
```

Parameter	Description
con	The Connection object.
tableName	The table name.
xmlName	The file name.
xmlData	The String with file data.

isXMLTable()

Checks if the table is an XML table, in which case returns TRUE.

Syntax

```
public boolean isXMLTable( java.sql.Connection con,
                           String tableName);
```

Parameter	Description
con	The Connection object.
tableName	The table name.

replaceXMLData()

Replaces text file as a row in XML table, returning `TRUE` if successful.

Syntax

```
public boolean replaceXMLData( java.sql.Connection con,
                               String tableName,
                               String xmlName,
                               String xmlData);
```

Parameter	Description
`con`	The Connection object.
`tableName`	The table name.
`xmlName`	The file name.
`xmlData`	The String with file data.

xmlTableExists()

Checks if the XML table exists, returning `TRUE` if successful.

Syntax

```
public boolean xmlTableExists( java.sql.Connection con,
                               String tableName);
```

Parameter	Description
`con`	The Connection object.
`tableName`	The table name.

DBAccessBeanInfo Class

This class provides information about the DBAccess Bean.

Syntax

```
public class DBAccessBeanInfo extends java.beans.SimpleBeanInfo
```

Table 10–26 Summary of Methods of DBAccessBeanInfo

Method	Description
DBAccessBeanInfo() on page 10-92	Class constructor.
getIcon() on page 10-92	Retrieves an image object representing the requested icon.
getPropertyDescriptors() on page 10-93	Retrieves an array of editable PropertyDescriptors.

DBAccessBeanInfo()

Class constructor.

Syntax

```
public  DBAccessBeanInfo();
```

getIcon()

Retrieves an image object representing the requested icon type for DBAcceess bean in toolbars, toolboxes, and so on.

Syntax

```
public java.awt.Image getIcon(int iconKind);
```

Parameter	Description
iconKind	The kind of icon requested.

getPropertyDescriptors()

Retrieves an array of DBAccess bean's editable PropertyDescriptors.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

XMLTransformPanel Class

XMLTransformPanel visual bean. Applies XSL transformations to XML documents and visualizes the result. Allows editing of input XML and XSL documents and files.

Syntax

```
public class XMLTransformPanel extends javax.swing.JPanel
```

XMLTransformPanel()

The class constructor. Creates an object of type `XMLTransformPanel`.

Syntax

```
public  XMLTransformPanel();
```

XMLTransformPanelBeanInfo Class

This class provides information about the XMLTransformPanel Bean.

Syntax

```
public class XMLTransformPanelBeanInfo extends java.beans.SimpleBeanInfo
```

Table 10–27 Summary of Methods of XMLTransformPanelBeanInfo

Method	Description
XMLTransformPanelBeanInfo() on page 10-95	Class constructor.
getIcon() on page 10-95	Retrieves an image object representing the requested icon type for XMLTransformPanel bean.
getPropertyDescriptors() on page 10-95	Retrieves an array of XMLTransformPanel bean's editable PropertyDescriptors.

XMLTransformPanelBeanInfo()

Class constructor.

Syntax

```
public  XMLTransformPanelBeanInfo();
```

getIcon()

Retrieves an image object that represents the icon type.

Syntax

```
public java.awt.Image getIcon( int iconKind);
```

Parameter	Description
iconKind	The kind of icon requested.

getPropertyDescriptors()

Retrieves an array of XMLTransformPanel bean's editable PropertyDescriptors.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

XMLTransViewer Class

Simple application that uses XMLTransformPanel. Can be used from the command line to edit and parse XML files, edit and apply XSL transformations and retrieve and save XML, XSL and result files in the file system or in the database.

Syntax

```
public class XMLTransViewer extends java.lang.Object
```

Table 10–28 Summary of Methods of XMLTransViewer

Method	Description
XMLTransViewer() on page 10-97	Class constructor
getReleaseVersion() on page 10-97	Returns the release version of the Oracle XML Transviewer, as a String.
main() on page 10-97	Starts a new XMLTransViewer.

XMLTransViewer()

Class constructor

Syntax

```
public  XMLTransViewer();
```

getReleaseVersion()

Returns the release version of the Oracle XML Transviewer, as a String.

Syntax

```
public static java.lang.String getReleaseVersion();
```

main()

The main function which starts a new XMLTransViewer.

Syntax

```
public static void main( String[] args);
```

Parameter	Description
`args`	Arguments of an XMLTransViewer instance.

oracle.xml.treeviewer Package

This is a visual bean. It displays XML documents as a tree, illustrating the DOM tree structure of a XML document. The user can collapse or expand the nodes.

The classes of the `oracle.xml.treeviewer` as summarized in Table 10–29.

Table 10–29 Summary of Classes of oracle.xml.transviewer

Class	Description
XMLTreeView Class on page 10-100	Shows an XML document as a tree.
XMLTreeViewBeanInfo Class on page 10-103	This class provides information about the XMLTreeView Bean.

XMLTreeView Class

Shows an XML document as a tree. Recognizes the following XML DOM nodes: `Tag, Attribute Name, Attribute Value, Comment, CDATA, PCDATA, PI Data, PI Name` and `NOTATION` Symbol. Takes as input an `org.w3c.dom.Document` object.

Syntax

```
public class XMLTreeView extends javax.swing.JPanel
```

Table 10–30 Fields of XMLTreeView

Field	Syntax	Description
`model`	`protected oracle.xml.treeviewer.XMLTreeModel model`	Data model for JTree formed from DOM nodes.
`scrollPane`	`protected transient javax.swing.JScrollPane scrollPane`	Container that displays the Tree. Supports horizontal and vertical scrolling.
`theTree`	`protected transient javax.swing.JTree theTree`	Java component to render the various nodes of the DOM Tree.

Table 10–31 Summary of Methods of XMLTreeView

Method	Description
XMLTreeView() on page 10-101	The class constructor.
getPreferredSize() on page 10-101	Returns the XMLTreeView preferred size.
getTree() on page 10-101	Returns the visual tree as a JTree.
getXMLTreeModel() on page 10-101	Returns the datamodel for JTree as XMLTreeModel.
setXMLDocument() on page 10-101	Associates the XMLTreeViewer with a XML document.
updateUI() on page 10-101	Forces the XMLTreeView to update/refresh UI.

XMLTreeView()

The class constructor. Creates an object of type `XMLTreeView`.

Syntax

```
public  XMLTreeView();
```

getPreferredSize()

Returns the Dimension object containing the XMLTreeView preferred size. Overrides `getPreferredSize()` in `javax.swing.JComponent` class.

Syntax

```
public java.awt.Dimension getPreferredSize();
```

getTree()

Returns the visual tree as a JTree.

Syntax

```
protected javax.swing.JTree getTree();
```

getXMLTreeModel()

Returns the datamodel for JTree as XMLTreeModel.

Syntax

```
protected oracle.xml.treeviewer.XMLTreeModel getXMLTreeModel();
```

setXMLDocument()

Associates the XMLTreeViewer with a XML document.

Syntax

```
public void setXMLDocument( org.w3c.dom.Document document);
```

Parameter	Description
`doc`	The Document to display.

updateUI()

Forces the XMLTreeView to update/refresh UI.

Syntax

```
public void updateUI();
```

XMLTreeViewBeanInfo Class

This class provides information about the XMLTreeView Bean.

Syntax

```
public class XMLTreeViewBeanInfo extends java.beans.SimpleBeanInfo
```

Table 10–32 Summary of Methods of XMLTreeViewBeanInfo

Method	Description
XMLTreeViewBeanInfo() on page 10-103	Class constructor.
getIcon() on page 10-103	Retrieves an image object representing the requested icon type for XMLTreeView.
getPropertyDescriptors() on page 10-103	Retrieves an array of XMLTreeViewl bean's editable PropertyDescriptors.

XMLTreeViewBeanInfo()

Class constructor.

Syntax

```
public  XMLTreeViewBeanInfo();
```

getIcon()

Retrieves an image object representing the requested icon type for XMLTreeView.

Syntax

```
public java.awt.Image getIcon( int iconKind);
```

Parameter	Description
iconKind	The kind of icon requested.

getPropertyDescriptors()

Retrieves an array of XMLTreeViewl bean's editable PropertyDescriptors.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

oracle.xml.differ Package

The `oracle.xml.differ` is a non-visual bean with a visual demo. It enables comparisons of two XML documents and generation of the differences as XSLT code. The XSLT can be applied to the first file to transform it into the second file. The visual demo enables the user to view the differences between the two XML documents graphically.

The classes of the `oracle.xml.differ` as summarized in Table 10–33.

Table 10–33 Summary of Classes of oracle.xml.differ

Class	Description
XMLDiff Class on page 10-106	Defines an interface for comparing two XML files.
XMLDiffBeanInfo Class on page 10-115	This class provides information about the XMLDiff Bean.

XMLDiff Class

Defines an interface for comparing two XML files. It enables two XML files to be compared to check for their equivalence. It provides the objects to display the differences, if any, in a graphical format. The differences can also be represented as XSL. The corresponding XSL stylesheet with the differences can be generated as a file or an XMLDocument object. The first XML file can be transformed into the second XML file by using the XSL stylesheet generated.

Syntax

```
oracle.xml.differ
```

Table 10–34 Summer of Methods of XMLDiff

Method	Description
XMLDiff() on page 10-107	Class constructor.
setFiles() on page 10-108	Sets the XML files which need to be compared.
setDocuments() on page 10-108	Sets the XML documents which need to be compared.
setInput1() on page 10-108	Sets the first XML file or document which need to be compared.
setInput2() on page 10-109	Sets the second XML file or document which need to be compared.
getDocument1() on page 10-110	Gets the document root as an XMLDocument object of the first XML tree.
getDocument2() on page 10-110	Gets the document root as an XMLDocument object of the second XML tree.
diff() on page 10-110	Finds the differences between the two XML files or the two XMLDocument objects. Retrieves the visual text panel as JTextPane object which visually shows the differences in the first XML file or document.
getDiffPane1() on page 10-110	Retrieves the visual text panel as JTextPane object which visually shows the differences in the first XML file or document.
getDiffPane2() on page 10-110	Retrieves the visual text panel as JTextPane object which visually shows the differences in the second XML file or document.

Table 10–34 (Cont.) Summer of Methods of XMLDiff

Method	Description
setIndentIncr() on page 10-111	Sets the indentation for the XSL generation.
setNewNodeIndentIncr() on page 10-111	Generates an XSL file, with user-defined filename, which represents the differences between the two initially set XML files.
generateXSLFile() on page 10-111	Generates an XSL file, with user-defined filename, which represents the differences between the two initially set XML files.
generateXSLDOC() on page 10-112	Generates an XSL stylesheet as an XMLDocument which represents the differences between the two initially set XML documents.
equals() on page 10-112	Compares two nodes. It is called by the differ algorithm. If needed, this function can be overwritten for customized comparisons.
domBuilderErrorCalled() on page 10-113	Method implementing the DOMBuilderErrorListener interface called only by the DOM parser when there is an error while parsing.
domBuilderError() on page 10-113	Method implementing the DOMBuilderErrorListener interface called only by the DOM parser.
domBuilderOver() on page 10-113	Method implementing DOMBuilderListener interface called only by a DOM parser thread when parsing completes.
domBuilderStarted() on page 10-114	Method implementing DOMBuilderListener interface called only by the DOM parser when the parsing starts.
printDiffTree() on page 10-114	Prints the differences tree which contains the node names and values which have been identified as differences by the algorithm. Useful for debugging.
setNoMoves() on page 10-114	Assume that there are no moves to be detected by the diff algorithm. This function should be called before the diff() function. It will result in a performance gain.

XMLDiff()

Class constructor.

Syntax

```
public XMLDiff();
```

setFiles()

Sets the XML files which need to be compared. Both files are parsed into DOM trees for comparison. This is faster than calling setInput1() and setInput2(). Throws the following exceptions:

- `java.io.IOException` when an I/O error occurs.
- `XMLParseException` when parsing XML document.
- `SAXException` when parsing XML document.
- `java.lang.InterruptedException` if a sleeping thread is interrupted.

Syntax

```
public void setFiles( java.io.File file1,
                      java.io.File file2);
```

Parameter	Description
`file1`	First XML file.
`file2`	Second XML file.

setDocuments()

Sets the XML documents which need to be compared.

Syntax

```
public void setDocuments( XMLDocument doc1,
                          XMLDocument doc2);
```

Parameter	Description
`doc1`	First XML document.

setInput1()

Sets the first XML file or document which need to be compared. The input file is parsed into a DOM tree for comparison. Throws the following exceptions:

- `java.io.IOException` when an I/O error occurs.

- `XMLParseException` when parsing XML document.
- `SAXException` when parsing XML document.
- `java.lang.InterruptedException` if a sleeping thread is interrupted.

The options are described in the following table.

Syntax	Description
`public void setInput1( File file1);`	Sets the first XML file which needs to be compared.
`public void setInput1( XMLDocument doc1);`	Sets the first XML document which needs to be compared.

Parameter	Description
`file1`	The first XML file.
`doc1`	The first XML document.

setInput2()

Sets the second XML file or document which need to be compared. The input file is parsed into a DOM tree for comparison. Throws the following exceptions:

- `java.io.IOException` when an I/O error occurs.
- `XMLParseException` when parsing XML document.
- `SAXException` when parsing XML document.
- `java.lang.InterruptedException` if a sleeping thread is interrupted.

The options are described in the following table.

Syntax	Description
`public void setInput2( File file2);`	Sets the second XML file which needs to be compared.
`public void setInput2( XMLDocument doc2);`	Sets the second XML document which needs to be compared.

Parameter	Description
file2	The second XML file.
doc2	The second XML document.

getDocument1()

Gets the document root as an XMLDocument object of the first XML tree.

Syntax

```
public XMLDocument getDocument1();
```

getDocument2()

Gets the document root as an XMLDocument object of the second XML tree.

Syntax

```
public XMLDocument getDocument2();
```

diff()

Finds the differences between the two XML files or the two XMLDocument objects. Returns `FALSE` if the XML files or docs are same, `TRUE` if they are different. Throws `java.lang.NullPointerException` when XML files are not parsed successfully, or if xml documents have not been set.

Syntax

```
public boolean diff();
```

getDiffPane1()

Retrieves the visual text panel as JTextPane object which visually shows the differences in the first XML file or document.

Syntax

```
public javax.swing.JTextPane getDiffPane1();
```

getDiffPane2()

Retrieves the visual text panel as JTextPane object which visually shows the differences in the second XML file or document.

Syntax

```
public javax.swing.JTextPane getDiffPane2();
```

setIndentIncr()

Sets the indentation for the XSL generation. This should be called before the `generateXSLFile()` or `generateXSLDoc()`. The indentation will be applied to all attributes only. For indenting newly inserted nodes, see `setNewNodeIndentIncr()`.

Syntax

```
public void setIndentIncr( int spaces);
```

Parameter	Description
`spaces`	Indentation increment in number of spaces for attributes.

setNewNodeIndentIncr()

Sets the indentation for the XSL generation. This should be called before the generateXSLFile() or generateXSLDoc(). The indentation will be applied to all newly inserted nodes only. For attributes indentation support, see `setIndentIncr()`.

Syntax

```
public void setNewNodeIndentIncr( int spaces);
```

Parameter	Description
`spaces`	Indentation increment in number of spaces for new nodes.

generateXSLFile()

Generates an XSL file, with user-defined filename, which represents the differences between the two initially set XML files. If the input filename is `NULL`, a default XSL file named `XMLDiff.xsl` will be generated. The first XML file can be transformed into the second XML file using the XSL stylesheet generated. If the XML are the same, the XSL generated will transform the first XML file into the second XML file, making the two files equivalent. Throws `java.io.IOException` if the XSL file is not created successfully.

Syntax

```
public void generateXSLFile( String fileName);
```

Parameter	Description
fileName	Output XSL file name.

generateXSLDOC()

Generates an XSL stylesheet as an XMLDocument which represents the differences between the two initially set XML documents. If the input filename is `NULL`, a default XSL file named `XMLDiff.xsl` will be generated. The first XML file can be transformed into the second XML file using the XSL stylesheet generated. If the XML are the same, the XSL generated will transform the first XML file into the second XML file, making the two files equivalent. Throws the following exceptions:

- `java.io.IOException` when an I/O error occurs.
- `java.io.FileNotFoundException` when the XSL file generated not found.
- `SAXException` when parsing XML document.
- `java.lang.InterruptedException` if a sleeping thread is interrupted.

Syntax

```
public void generateXSLDoc();
```

equals()

Compares two nodes. It is called by the differ algorithm. If needed, this function can be overwritten for customized comparisons.

Syntax

```
protected boolean equals( Node node1,
                         Node node2);
```

Parameter	Description
node1	The first node to compare.
node2	The second node to compare.

domBuilderErrorCalled()

Method implementing the DOMBuilderErrorListener interface called only by the DOM parser when there is an error while parsing. Specified by domBuilderErrorCalled in DOMBuilderErrorListener Interface.

Syntax

```
public void domBuilderErrorCalled( DOMBuilderErrorEvent p0);
```

Parameter	Description
p0	Error object thrown by the parser.

domBuilderError()

Method implementing the DOMBuilderErrorListener interface called only by the DOM parser. Specified by domBuilderError in DOMBuilderListener interface.

Syntax

```
public void domBuilderError( DOMBuilderEvent p0);
```

Parameter	Description
p0	Parser event errors handled by domBuilderErrorCalled().

domBuilderOver()

Method implementing DOMBuilderListener interface called only by a DOM parser thread when parsing completes. Specified by domBuilderOver in DOMBuilderListener interface

Syntax

```
public void domBuilderOver ( DOMBuilderEvent p0);
```

Parameter	Description
p0	Parser event.

domBuilderStarted()

Method implementing DOMBuilderListener interface called only by the DOM parser when the parsing starts. Specified by domBuilderStarted in DOMBuilderListener interface.

Syntax

```
public void domBuilderStarted( DOMBuilderEvent p0);
```

Parameter	Description
p0	Parser event.

printDiffTree()

Prints the diff tree which contains the node names and values which have been identified as different by the algorithm. Useful for debugging. Throws `java.io.IOExceptio`n if the XSL file is not created successfully.

Syntax

```
public void printDiffTree( int tree,
                           BufferedWriter out);
```

Parameter	Description
tree	The tree to print, 1 or 2.
out	The BufferredWriter containing the printed diff tree.

setNoMoves()

Assume that there are no moves to be detected by the diff algorithm. This function should be called before the diff() function. It will result in a performance gain.

Syntax

```
public void setNoMoves();
```

XMLDiffBeanInfo Class

This class provides information about the XMLDiff Bean.

Syntax

```
public class XMLDiffBeanInfo extends java.beans.SimpleBeanInfo
```

Table 10–35 Summary of Methods of XMLDiffBeanInfo

Method	Description
XMLDiffBeanInfo() on page 10-115	Class constructor.
getPropertyDescriptors() on page 10-115	Retrieves an image object representing the requested icon type for XMLDiff bean in toolbars, toolboxes, and so on.
getIcon() on page 10-115	Retrieves an array of XMLDiff bean's editable PropertyDescriptors

XMLDiffBeanInfo()

Class constructor.

Syntax

```
public XMLDiffBeanInfo();
```

getPropertyDescriptors()

Retrieves an array of XMLDiff bean's editable PropertyDescriptors. Overrides `getPropertyDescriptors()` in `java.beans.SimpleBeanInfo` Class.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

getIcon()

Retrieves an image object representing the requested icon type for XMLDiff bean in toolbars, toolboxes, and so on. Overrides `getIcon()` in `java.beans.SimpleBeanInfo` class.

Syntax

```
public java.awt.Image getIcon( int iconKind);
```

Parameter	Description
iconKind	The kind of icon requested.

11

Compression for Java

Because XML files typically have repeated tags, compression of a streaming of XML structure and tokenizing of the XML tags can be very effective. Since it parses documents in real time, without waiting for the complete document to be read first, compression and decompression yield high performance benefits when exchanging a document over the internet between two sub-systems. XML compression retains the structural and hierarchical information of the XML data in the compressed format.

This chapter describes the `oracle.xml.parser.v2` package classes responsible for compression:

- CXMLHandlerBase Class
- CXMLParser Class

See Also:

- *Oracle Application Developer's Guide - XML*

CXMLHandlerBase Class

The SAX compression is implemented using SAX Handler, which compresses the data based on SAX events. To use the SAX compression, the application needs to implement this interface and register with the SAX parser through the `Parser.setDocumentHandler()`.

Syntax

```
public class CXMLHandlerBase implements oracle.xml.parser.v2.XMLDocumentHandler
```

Table 11–1 Summary of Methods of CXMLHandlerBase

Method	Description
CXMLHandlerBase() on page 11-3	Creates a new CXMLHandlerBase.
cDATASection() on page 11-4	Receives notification of a CDATA Section.
characters() on page 11-4	Receives notification of Character Data inside an element.
comment() on page 11-4	Receives notification of Comment.
endDoctype() on page 11-5	Receives notification of end of the DTD.
endDocument() on page 11-5	Receives notification of end of the Document.
endElement() on page 11-5	Receives notification of end of the Element.
endPrefixMapping() on page 11-5	Receives notification of end of scope of Prefix Mapping.
getCXMLContext() on page 11-6	Returns the CXML Context used for compression.
getProperty() on page 11-6	Looks up and returns the value of a property.
ignorableWhitespace() on page 11-6	Receives notification of ignorable whitespace in element content.
processingInstruction() on page 11-7	Receives notification of a processing instruction.
setDoctype() on page 11-7	Sets DTD to receive notification of that DTD.
setDocumentLocator() on page 11-7	Sets Locator to receive that Locator object for the document event.

Table 11–1 (Cont.) Summary of Methods of CXMLHandlerBase

Method	Description
setError() on page 11-8	Sets XMLError handler to receive notification of that XMLError handler.
setProperty() on page 11-8	Sets the value of a property.
setTextDecl() on page 11-8	Sets Text XML Declaration to receive notification of that Text XML Declaration.
setXMLDecl() on page 11-9	Sets XML Declaration to receive notification of that XML Declaration.
setXMLSchema() on page 11-9	Sets XMLSchema to receive notification of that XMLSchema object.
skippedEntity() on page 11-10	Receives notification of a skipped entity.
startDocument() on page 11-10	Receives notification of the beginning of the document.
startElement() on page 11-10	Receives notification of the beginning of an element.
startPrefixMapping() on page 11-11	Receives notification of the beginning of the scope of prefix URI mapping.

CXMLHandlerBase()

Creates a new CXMLHandlerBase. The options are described in the following table.

Syntax	Description
`public CXMLHandlerBase();`	Default Constructor. Creates a new CXMLHandlerBase.
`public CXMLHandlerBase( ObjectOutput out);`	Constructs the CXMLHandlerBase using the ObjectOutput stream
`public CXMLHandlerBase( oracle.xml.io.XMLObjectOutput out);`	Constructs the CXMLHandlerBase using the XMLObjectOutputStream.

Parameter	Description
`out`	The output stream.

cDATASection()

Receives notification of a CDATA Section. The Parser will invoke this method once for each CDATA Section found. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Syntax

```
public void cDATASection( char[] cbuf,
                          int start,
                          int len);
```

Parameter	Description
`cbuf`	The CDATA section characters.
`start`	The start position in the character array.
`len`	The number of characters to use from the character array.

characters()

Receives notification of character data inside an element.

Syntax

```
public void characters( char[] cbuf,
                        int start,
                        int len);
```

Parameter	Description
`cbuf`	Characters
`start`	The starting position in the character array.
`len`	The number of characters to use from the character array.

comment()

Receives notification of a comment. The Parser will invoke this method once for each comment found: note that comment may occur before or after the main

document element. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Syntax

```
public void comment( String text);
```

Parameter	Description
`text`	The comment data; `NULL` if none is supplied.

endDoctype()

Receives notification of end of the DTD. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Syntax

```
public void endDoctype();
```

endDocument()

Receives notification of the end of the document. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Syntax

```
public void endDocument();
```

endElement()

Receives notification of the end of the element.

Syntax

```
public void endElement( oracle.xml.parser.v2.NSName elem);
```

Parameter	Description
`elem`	The element.

endPrefixMapping()

Receives notification of the end of scope of prefix URI mapping.

Syntax

```
public void endPrefixMapping( String prefix);
```

Parameter	Description
prefix	The prefix that was being mapped.

getCXMLContext()

Returns the CXML Context used for compression.

Syntax

```
public oracle.xml.comp.CXMLContext getCXMLContext();
```

getProperty()

Looks up and returns the value of a property. The property name is any fully-qualified URI.

Syntax

```
public java.lang.Object getProperty( String name);
```

Parameter	Description
name	The property name, which is a fully-qualified URI.

ignorableWhitespace()

Receives notification of ignorable whitespace in element content.

Syntax

```
public void ignorableWhitespace( char[] cbuf,
                                 int start,
                                 int len);
```

Parameter	Description
cbuf	The Character from XML document.
start	The start position in the array.

Parameter	Description
len	The number of characters to read from the array.

processingInstruction()

Receives notification of a processing instruction.

Syntax

```
public void processingInstruction( String target,
                                   String data);
```

Parameter	Description
target	The processing instruction target.
data	The processing instruction data.

setDoctype()

Registers DTD so can subsequently receive notification on that DTD. The Parser will invoke this method after calling `startDocument()` to register the DTD used. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setDoctype( oracle.xml.parser.v2.DTD dtd);
```

Parameter	Description
dtd	The DTD node.

setDocumentLocator()

Registers Locator object so can subsequently receive notification for that Locator object for the document event. By default, do nothing. This method could be overridden by the subclasses events.

Syntax

```
public void setDocumentLocator( org.xml.sax.Locator locator);
```

Parameter	Description
locator	The locator for the SAX document events.

setError()

Registers XMLError handler so can subsequently receive notification of that XMLError handler. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setError( oracle.xml.parser.v2.XMLError he);
```

Parameter	Description
he	The XMLError object.

setProperty()

Sets the value of a property. The property name is any fully-qualified URI.

Syntax

```
public void setProperty( String name,
                         Object value);
```

Parameter	Description
name	The property name, which is a fully-qualified URI.
value	The requested value for the property.

setTextDecl()

Registers Text XML Declaration so can subsequently receive notification of that Text XML Declaration. The Parser will invoke this method once for each text `XMLDecl`.

Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setTextDecl( String version,
                         String encoding);
```

Parameter	Description
`version`	The version number; `NULL` if not specified.
`encoding`	The encoding name.

setXMLDecl()

Registers XML Declaration so can subsequently receive notification of that XML Declaration. The Parser will invoke this method once for `XMLDecl`. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setXMLDecl( String version,
                        String standalone,
                        String encoding);
```

Parameter	Description
`version`	The version number.
`standalone`	The standalone value; `NULL` if not specified.
`encoding`	The encoding name; `NULL` if not specified.

setXMLSchema()

Registers XMLSchema so can subsequently receive notification of that XMLSchema object. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Syntax

```
public void setXMLSchema( Object s);
```

Parameter	Description
s	The XMLSchema object.

skippedEntity()

```
public void skippedEntity( String name);
```

Syntax

Receives 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 `external-general-entities` and the `external-parameter-entities` properties. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Parameter	Description
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]".

startDocument()

Receives notification of the beginning of the document. By default, do nothing. This method could be overridden by the subclasses to take specific actions at the beginning of a document.

Syntax

```
public void startDocument();
```

startElement()

Receives notification of the beginning of an element.

Syntax

```
public void startElement( oracle.xml.parser.v2.NSName elem,
                          oracle.xml.parser.v2.SAXAttrList attributes);
```

Parameter	Description
elem	The element.
attributes	Attributes of the element.

startPrefixMapping()

Receives notification of the beginning of the scope of prefix URI mapping.

Syntax

```
public void startPrefixMapping( String prefix,
                                String uri);
```

Parameter	Description
prefix	The Namespace prefix being declared.
uri	The Namespace URI to which the prefix is mapped.

CXMLParser Class

This class implements the re-generation of XML document from the compressed stream by generating the SAX events from them.

Syntax

```
public class CXMLParser
```

Table 11–2 Summary of Methods of CXMLHandlerBase

Method	Description
startPrefixMapping() on page 11-11	Creates a new XML Parser object for reading the compressed stream.
CXMLParser() on page 11-12	Retrieves content handler.
getContentHandler() on page 11-12	Retrieves the current error handler.
getErrorHandler() on page 11-13	Parses the compressed stream and generates the SAX events.
parse() on page 11-13	Parses the compressed stream and generates the SAX events.
setContentHandler() on page 11-13	Registers a content event handler.
setErrorHandler() on page 11-13	Registers an error event handler.

CXMLParser()

Creates a new XML Parser object for reading the compressed stream.

Syntax

```
public  CXMLParser();
```

getContentHandler()

Retrieves content handler; returns `NULL` if the handler has not been registered.

Syntax

```
public org.xml.sax.ContentHandler getContentHandler();
```

getErrorHandler()

Retrieves the current error handler; returns `NULL` if the handler has not been registered.

Syntax

```
public org.xml.sax.ErrorHandler getErrorHandler();
```

parse()

Parses the compressed `SAXException`.

- `IOException` - an error due to I/O operations

Syntax

```
public void parse( String inFile);
```

Parameter	Description
`inFile`	The input source which needs to be parsed to regenerate the SAX events.

setContentHandler()

Registers a content event handler.

Syntax

```
public void setContentHandler( org.xml.sax.ContentHandler handler);
```

Parameter	Description
`handler`	The content handler

setErrorHandler()

Registers an error event handler.

Syntax

```
public void setErrorHandler( org.xml.sax.ErrorHandler handler);
```

Parameter	Description
handler	The error handler

12

Simple Object Access Protocol (SOAP)

Oracle SOAP is an implementation of the Simple Object Access Protocol. Oracle SOAP is based on the SOAP open source implementation developed by the Apache Software Foundation.

SOAP is a transport protocol for sending and receiving requests and responses across the Internet. It is based on XML and HTTP. SOAP is transport protocol-independent and operating system-independent. It provides the standard XML message format for all applications. SOAP uses the XML Schema standard of the World Wide Web Consortium (W3C).

Oracle SOAP APIs are contained in these packages:

- oracle.soap.server Package
- oracle.soap.transport Package
- oracle.soap.transport.http Package
- oracle.soap.util.xml Package

See Also:

- `http://www.w3.org/TR/SOAP/`
- `http://xml.apache.org/soap/`
- *Oracle Application Developer's Guide - XML*

oracle.soap.server Package

Package oracle.soap.server contains the interfaces and classes that implement the API for SOAP administrative clients and the provider implementation for Java classes. These include the Service Manager and the Provider Manager. These administrative clients are services that support dynamic deployment of new services and new providers.

Table 12–1 lists the interfaces and classes that provide support for Oracle SOAP in the XDK for Java.

Table 12–1 Summary of Classes and Interfaces of oracle.soap.server

Class/Interface	Description
Handler Interface on page 12-3	Defines the interface for a pluggable handler in the SOAP server.
Provider Interface on page 12-7	Defines the capabilities that must be supported for each type of service provider.
ProviderManager Interface on page 12-10	Defines the Provider Manager used by the SOAP engine to deploy providers, undeploy providers, and access provider deployment information.
ServiceManager Interface on page 12-14	Defines the Service Manager used by the SOAP engine to deploy services, undeploy services, and to access service deployment information.
ContainerContext Class on page 12-17	Defines the context of the container in which the SOAP server is running
Class Logger on page 12-21	Defines the capabilities that must be supported by a logger implementation; the logger is used to persistently record error and informational messages.
ProviderDeploymentDescriptor Class on page 12-26	Defines the deployment information for a specific provider.
RequestContext Class on page 12-30	Defines all of the context for a SOAP request, including information that is passed to the provider and information that the provider must set before returning.
ServiceDeploymentDescriptor Class on page 12-37	Defines the deployment information for a SOAP service, independent of its provider type.
UserContext Class on page 12-51	Defines the user context for a SOAP service request.

Handler Interface

`Handler` defines the interface for a pluggable handler in the SOAP server. This class does not imply any policies about when the handler in invoked. A handler implementation must provide a no-args constructor and be thread-safe.

Syntax

```
public interface Handler
```

Table 12–2 Fields of Handler

Field	Syntax	Description
`REQUEST_TYPE`	`public static final int REQUEST_TYPE`	Handler invocation is part of request chain.
`RESPONSE_TYPE`	`public static final int RESPONSE_TYPE`	Handler invocation is part of response chain.
`ERROR_TYPE`	`public static final int ERROR_TYPE`	Handler invocation is part of error chain.

Table 12–3 Summary of Methods of Handler

Method	Description
destroy() on page 12-4	Cleans-up handler (one time only).
getName() on page 12-4	Returns this handler's name.
getOptions() on page 12-4	Returns options that are specific to the handler implementation.
init() on page 12-4	Initializes handler (one-time only).
invoke() on page 12-5	Invokes the requested handler as part of the specified chain type.
setName() on page 12-5	Sets the name of the handler. This method must be called before init().
setOptions() on page 12-6	Sets the options for the handler for subsequent use by init.

destroy()

Cleans-up handler (one time only). This method will be invoked by the SOAP server exactly once before the server shuts down. This gives the handler the opportunity to do cleanup of global state. Throws `SOAPException` if unable to destroy.

Syntax

```
public abstract void destroy();
```

getName()

Returns this handler's name.

Syntax

```
public abstract String getName();
```

getOptions()

Returns options that are specific to the handler implementation.

Syntax

```
public abstract Properties getOptions();
```

init()

Initializes handler (one-time only). This method will be invoked by the SOAP server exactly once before the server makes any invocations on the handler, allowing the handler to set up any global state. It uses any options that were set previously through `setOptions()`. Throws `SOAPException` if unable to initialize the handler.

Syntax

```
public abstract void init( SOAPServerContext ssc);
```

Parameter	Description
`ssc`	The SOAP server context, which contains the logger for informational messages.

invoke()

Invokes the requested handler as part of the specified chain type. Note that execution of a chain of request handlers or response handlers will terminate immediately if any handler throws a `SOAPException`. In contrast, all handlers in an error chain will be invoked, regardless of whether or not any handler throws an exception. In the case of an exception in an error handler, the exception is logged and discarded. Throws `SOAPException` if handler invocation failed.

Syntax

```
public abstract void invoke( int chainType, RequestContext requestContext);
```

Parameter	Description
`chainType`	`The following chainTypes are supported:` `Handler.REQUEST_TYPE` if the handler is being invoked as part of a request chain, before the service is invoked `Handler.RESPONSE_TYPE` if the handler is being invoked as part of a response chain, after the service has been invoked `Handler.ERROR_TYPE` if the handler is being invoked as part of an error chain, in case of an error during any one of request chain, service invocation, or response chain
`requestContext`	The relevant request context.

setName()

Sets the name of the handler. This method must be called before `init()`.

Syntax

```
public abstract void setName( String name);
```

Parameter	Description
`name`	The name of the handler instance.

setOptions()

Sets the options for the handler for subsequent use by init. This method must be called before `init()`.

Syntax

```
public abstract void setOptions( Properties options);
```

Parameter	Description
`options`	Options that are specific to the handler implementation.

Provider Interface

Provider defines the capabilities that must be supported for each type of service provider, such as Java class or stored procedure. Providers are responsible for service authorization, and parameter unmarshalling/marshalling.

Providers, aka provider instances, must be deployed to the SOAP handler. Each provider deployment must define the provider name, Java classname that implements the provider (which must be an implementation of this interface), and any number of provider-specific key-value pairs. Given the provider deployment information, the SOAP handler will interact with the providers solely through this interface.

The SOAP handler will create one instance for each deployed provider instance. It is possible to have one or more instances of each provider implementation (which is not to say that is necessarily recommended). In any event, each instance of a provider must be able to handle requests concurrently.

A provider implementation must provide a no-args constructor and be thread-safe

Syntax

```
public interface Provider
```

Table 12–4 Summary of Methods in Provider

Method	Description
destroy() on page 12-7	Cleans up provider instance (one time only).
getId() on page 12-8	Returns this providers name, which is unique within the SOAP handler.
init() on page 12-8	Initializes provider instance (one time only).
invoke() on page 12-8	Invokes the requested method in the specified service, where the SOAP request is completely described in the request context.

destroy()

Cleans up provider instance (one time only). This method will be invoked by the SOAP handler exactly once before the handler shuts down. This gives the provider

the opportunity to do cleanup of provider-global state. Throws `SOAPException` if unable to destroy.

Syntax

```
public abstract void destroy();
```

getId()

Returns this providers name, which is unique within the SOAP handler.

Syntax

```
public abstract String getId();
```

init()

Initializes provider instance (one time only). This method will be invoked by the SOAP handler exactly once before the handler makes any requests to services supported by the provider, allowing the provider to set up any provider-global context. Throws `SOAPException` if unable to initialize and therefore unable to provide services.

Syntax

```
public abstract void init(ProviderDeploymentDescriptor pd,
                          SOAPServerContext ssc);
```

Parameter	Description
`pd`	The provider descriptor which contains the provider deployment information.
`ssc`	The SOAP server context that h contains the logger for informational messages.

invoke()

Invokes the requested method in the specified service, where the SOAP request is completely described in the request context. Throws `SOAPException` if error during method invocation for any number of reasons, including user does not have permission, method does not exist.

Syntax

```
public abstract void invoke( RequestContext requestContext);
```

Parameter	Description
`requestContext`	`RequestContext` that contains information used to process the request.

ProviderManager Interface

Provider Manager defines the interface to manage providers. The provider manager is used by the SOAP engine to deploy providers, undeploy providers, and access provider deployment information. The provider manager may cache deployment information and is responsible to maintain the cache.

The HTTP server provides security for the provider manager. The provider manager can be configured with a URL that requests must be made to in order for the request to be accepted. If a SOAP request for the provider manager is made to any other URL, the request will be rejected. This URL should be an alias to the SOAP servlet, and HTTP security can be set to control which users can post to the URL.

Syntax

```
public interface ProviderManager
```

Table 12–5 Summary of Methods in ProviderManager

Method	Description
deploy() on page 12-10	Deploys the given provider.
destroy() on page 12-11	Cleans up the provider manager.
getRequiredRequestURI() on page 12-11	Returns the URI that provider manager requests.
init() on page 12-11	Initializes the provider manager.
list() on page 12-11	Returns an array of provider ids for all providers that have been deployed.
query() on page 12-12	Returns the deployment descriptor for the given provider.
setServiceManager() on page 12-12	Makes the service manager available to the provider manager.
undeploy() on page 12-12	Undeploys the given provider.

deploy()

Deploys the given provider. Throws `SOAPException` if unable to deploy.

Syntax

```
public abstract void deploy(ProviderDeploymentDescriptor providerId);
```

Parameter	Description
providerId	The id of the provider to deploy.

destroy()

Cleans up the provider manager. Throws SOAPException if unable to cleanup the provider manager.

Syntax

```
public abstract void destroy();
```

getRequiredRequestURI()

Returns the URI that provider manager requests, or NULL if any URI can be used. Request must be made to in order to be accepted. Requests made to any other URI must be rejected.

Syntax

```
public abstract String getRequiredRequestURI();
```

init()

Initializes the provider manager. Throws SOAPException if unable to access the deployment information.

Syntax

```
public abstract void init(Properties options);
```

Parameter	Description
options	The options required to setup access to the deployment information.

list()

Returns an array of provider ids for all providers that have been deployed. Throws SOAPException if unable to list provider ids.

Syntax

```
public abstract String[] list();
```

query()

Returns the deployment descriptor for the given provider. Throws `SOAPException` if the provider is not found.

Syntax

```
public abstract ProviderDeploymentDescriptor query( String providerId);
```

Parameter	Description
`providerId`	The id of the provider.

setServiceManager()

Makes the service manager that is being used to manage service deployment information available to the provider manager. The provider manager may use the service manager to ensure that a provider is not undeployed as long as any services are deployed under that provider.

Syntax

```
public abstract void setServiceManager( ServiceManager serviceManager);
```

Parameter	Description
`providerManager`	The provider manager that is managing provider deployment information for the SOAP server.

undeploy()

Undeploys the given provider, and returns its descriptor containing the deployment information for the provider that has been undeployed. Throws `SOAPException` if the provider is not found or failed to undeploy.

Syntax

```
public abstract ProviderDeploymentDescriptor undeploy( String providerId);
```

Parameter	Description
`providerId`	The id of the provider to undeploy.

ServiceManager Interface

Service Manager defines the interface to manage services. The Service Manager is used by the SOAP engine to deploy services, undeploy services, and to access service deployment information. The Service Manager may cache deployment information and is responsible for maintaining the cache.

The HTTP server provides security for the service manager. The service manager can be configured with a URL that requests must be made to in order for the request to be accepted. If a SOAP request for the service manager is made to any other URL, the request will be rejected. This URL should be an alias to the SOAP servlet, and HTTP security can be set to control which users can post to the specified URL.

Syntax

```
public interface ServiceManager
```

Table 12–6 Summary of Methods in ServiceManager

Method	Description
getRequiredRequestURI() on page 12-14	Returns the URI that service manager requests.
deploy() on page 12-15	Deploys the given service.
destroy() on page 12-15	Cleans up the service manager.
init() on page 12-15	Initializes the service manager.
list() on page 12-16	Returns an array of service ids for all services that have been deployed, regardless of the provider.
query() on page 12-16	Returns the deployment descriptor for the given service.
undeploy() on page 12-16	Undeploys the given service, and returns its descriptor.

getRequiredRequestURI()

Returns the URI that service manager requests, or `NUKLL` if any URI can be used. Requests must be made to in order to be accepted. Requests made to any other URI must be rejected.

Syntax

```
public abstract String getRequiredRequestURI();
```

deploy()

Deploys the given service. Throws `SOAPException` if unable to deploy.

Syntax

```
public abstract void deploy(ServiceDeploymentDescriptor sd);
```

Parameter	Description
`sd`	The service descriptor for the service to deploy.

destroy()

Cleans up the service manager. Throws `SOAPException` if unable to cleanup the service manager.

Syntax

```
public abstract void destroy();
```

init()

Initializes the service manager. The implementation should be able to handle a null value for the provider manager. Throws `SOAPException` if unable to access the service deployment information.

Syntax

```
public abstract void init( Properties options,
                           ProviderManager providerManager);
```

Parameter	Description
`options`	The options required to setup access to service deployment information.
`providerManager`	The provider manager that is managing provider deployment information for the SOAP server, or null if the provider manager is not supplied. The service manager may want to use the provider manager to confirm the existence of the provider when a new service is deployed.

list()

Returns an array of service ids for all services that have been deployed, regardless of the provider. Throws `SOAPException` if unable to list service ids.

Syntax

```
public abstract String[] list();
```

query()

Returns the deployment descriptor for the given service. Throws `SOAPException` if the service is not found.

Syntax

```
public abstract ServiceDeploymentDescriptor query( String serviceId);
```

Parameter	Description
`serviceId`	The unique URI of the service.

undeploy()

Undeploys the given service, and returns its descriptor. Throws `SOAPException` if the service is not found or failed to undeploy.

Syntax

```
public abstract ServiceDeploymentDescriptor undeploy( String serviceId);
```

Parameter	Description
`serviceId`	The URI of the service to undeploy.

ContainerContext Class

ContainerContext class defines the context of the container in which the SOAP server is running. The actual content depends on the environment in which the server is running, such as in a servlet engine. This class should contain only container-specific content.

Syntax

```
public class ContainerContext extends Object
```

Table 12–7 Fields of ContainerContext

Field	Syntax	Description
SERVLET_CONTAINER	public static final String SERVLET_CONTAINER	The value for a servlet container type.

Table 12–8 Summary of Methods of ContainerContext

Method	Description
ContainerContext() on page 12-18	Class constructor.
getAttribute() on page 12-18	Returns the attribute with the given name.
getAttributeNames() on page 12-18	Returns an Enumeration containing the attribute names available within this SOAP context.
getContainerType() on page 12-18	Returns the container type in which the SOAP server is running.
getHttpServlet() on page 12-18	Returns the HTTP servlet that is processing the SOAP request if the container type is SERVLET_CONTAINER.
removeAttribute() on page 12-19	Removes the attribute with the given name from the context.
setAttribute() on page 12-19	Binds an object to a given attribute name in this SOAP context.
setContainerType() on page 12-19	Sets the container type.
setHttpServlet() on page 12-20	Sets the HTTP servlet for a SOAP server running in a SERVLET_CONTAINER type of container.

ContainerContext()

Class constructor.

Syntax

```
public ContainerContext();
```

getAttribute()

Returns the attribute with the given name, or NULL if there is no attribute by that name.

Syntax

```
public Object getAttribute( String name);
```

Parameter	Description
name	A String specifying the name of the attribute.

getAttributeNames()

Returns an Enumeration containing the attribute names available within this SOAP context.

Syntax

```
public Enumeration getAttributeNames();
```

getContainerType()

Returns the container type in which the SOAP server is running.

Syntax

```
public String getContainerType();
```

getHttpServlet()

Returns the HTTP servlet processing SOAP request if the container type is SERVLET_CONTAINER, or NULL if the servlet attribute is not set.

Syntax

```
public HttpServlet getHttpServlet();
```

removeAttribute()

Removes the attribute with the given name from the context. After removal, subsequent calls to getAttribute(java.lang.String) to retrieve the attribute's value will return NULL.

Syntax

```
public void removeAttribute( String name);
```

Parameter	Description
name	A String specifying the name of the attribute to be removed.

setAttribute()

Binds an object to a given attribute name in this SOAP context. If the name specified is already used for an attribute, this method will remove the old attribute and bind the name to the new attribute. Neither the name nor the object may be NULL.

Syntax

```
public void setAttribute( String name, Object object);
```

Parameter	Description
name	A non-null String specifying the name of the attribute.
object	An non-null Object representing the attribute to be bound.

setContainerType()

Sets the container type.

Syntax

```
public void setContainerType( String containerType);
```

Parameter	Description
containerType	The type of container in which the SOAP server is running.

setHttpServlet()

Sets the HTTP servlet for a SOAP server running in a `SERVLET_CONTAINER` type of container.

Syntax

```
public void setHttpServlet(HttpServlet servlet);
```

Parameter	Description
`servlet`	The HttpServlet that is processing the SOAP request.

Class Logger

Logger defines the capabilities that must be supported by a logger implementation. The logger is used to persistently record error and informational messages.

Each log request specifies the severity, and the information should be logged iff the severity is at least as high as the specified severity.

The order of severity in increasing order is:

- SEVERITY_ERROR
- SEVERITY_STATUS
- SEVERITY_DEBUG

For example, if the severity is set to SEVERITY_STATUS, any log request with severity of either SEVERITY_STATUS or SEVERITY_ERROR will be logged.

Syntax

```
public abstract class Logger extends Object
```

Table 12–9 Fields of Logger

Field	Syntax	Description
SEVERITY_ERROR	public static final int SEVERITY_ERROR	Severity level for logging error messages.
SEVERITY_STATUS	public static final int SEVERITY_STATUS	Severity level for logging status messages.
SEVERITY_DEBUG	public static final int SEVERITY_DEBUG	Severity level for logging information for debugging purposes.
SEVERITY_INVALID	protected static final int SEVERITY_INVALID	Indicates an invalid severity setting.
SEVERITY_NAMES	public static String SEVERITY_NAMES[]	Printable names for each severity level, indexed by severity.
DEFAULT_SEVERITY	public static final int DEFAULT_SEVERITY	The default severity level setting for determining which log requests are actually logged. The default is SEVERITY_STATUS.

Table 12–9 (Cont.) Fields of Logger

Field	Syntax	Description
OPTION_SEVERITY	public static final String OPTION_SEVERITY	Configuration option that specifies the severity for the logger.
m_severity	protected int m_severity	The logger's severity setting.

Methods of Logger

Table 12–10 Summary of Methods of Logger

Method	Description
Logger() on page 12-22	Class constructor.
getSeverity() on page 12-22	Returns the current severity setting for the logger.
getSeverityName() on page 12-23	Returns the severity name associated with the given severity.
getSeverityValue() on page 12-23	Returns the severity value associated with the given severity name.
init() on page 12-23	Initializes of the logger (one-time only) with its configuration parameters.
isLoggable() on page 12-24	Determines if a message would be logged at the given severity level.
log() on page 12-24	Logs messages.
setSeverity() on page 12-25	Sets the current severity.

Logger()

Class constructor.

Syntax

```
public Logger();
```

getSeverity()

Returns the current severity setting for the logger.

Syntax

```
public int getSeverity();
```

getSeverityName()

Returns the severity name associated with the given severity.

Syntax

```
protected final String getSeverityName( int severity);
```

Parameter	Description
severity	The severity level (SEVERITY_xxx).

getSeverityValue()

Returns the severity value associated with the given severity name.

Syntax

```
protected final int getSeverityValue( String severityName);
```

Parameter	Description
severityName	The name of the severity level, such as error.

init()

Initializes of the logger (one-time only) with its configuration parameters. Throws SOAPException if unable to initialize the logger.

Syntax

```
public abstract void init( Properties options,
                           ContainerContext context);
```

Parameter	Description
options	The configuration options for the logger.
context	The context of the container in which the SOAP server is running, which includes information that may be used by the logger.

isLoggable()

Determines if a message would be logged at the given severity level. Returns `TRUE` if a message would be logged at the given severity level, `FALSE` otherwise.

Syntax

```
public boolean isLoggable( int severity);
```

Parameter	Description
`severity`	The severity level to check.

log()

Logs messages. The options are described in the following table.

Syntax	Description
`public abstract void log( String msg, int severity);`	Logs the given message at the given severity.
`public abstract void log( String msg, Throwable t, int severity);`	Logs the given message and exception at the given severity.
`public abstract void log( Throwable t, int severity);`	Logs the given exception at the given severity.

Parameter	Description
`msg`	The message to log.
`severity`	The severity at which to log the information.
`t`	The throwable exception to log.

setSeverity()

Sets the current severity.

Syntax

```
public void setSeverity(int severity);
```

Parameter	Description
severity	The new severity setting for the logger.

ProviderDeploymentDescriptor Class

`ProviderDeploymentDescriptor` defines the deployment information for a specific provider. Different providers may be deployed using the same implementation and be distinguished only by their provider descriptor.

Syntaxr

```
public final class ProviderDeploymentDescriptor extends Object implements
Serializable
```

Table 12–11 Summary of Methods of ProviderDeploymentDescriptor

Method	Descriptor
ProviderDeploymentDescriptor() on page 12-26	Constructs a new instance of a provider descriptor.
fromXML() on page 12-27	Builds and returns a provider descriptor from the given XML document.
getClassname() on page 12-27	Returns the name of the class that implements this provider.
getId() on page 12-27	Returns the unique id for this provider.
getOptions() on page 12-27	Returns the provider-specific options
getProviderType() on page 12-27	Returns the provider type.
setClassname() on page 12-28	Sets the name of the class that implements this provider.
setId() on page 12-28	Sets the provider id.
setOptions() on page 12-28	Sets the options.
setProviderType() on page 12-28	Sets the provider type.
toString() on page 12-29	Writes out the service deployment descriptor to String.
toXML() on page 12-29	Writes out the service deployment descriptor as XML.

ProviderDeploymentDescriptor()

Constructs a new instance of a provider descriptor.

Syntax

```
public ProviderDeploymentDescriptor();
```

fromXML()

Builds and returns a provider descriptor from the given XML document.

Syntax

```
public static ProviderDeploymentDescriptor fromXML( Element root);
```

Parameter	Description
root	The root of the document that represents the XML provider descriptor.

getClassname()

Returns the name of the class that implements this provider.

Syntax

```
public String getClassname();
```

getId()

Returns the unique id for this provider.

Syntax

```
public String getId();
```

getOptions()

Returns the provider-specific options, or value pairs that represent the provider-specific options for this service.

Syntax

```
public Hashtable getOptions();
```

getProviderType()

Returns the provider type.

Syntax

```
public String getProviderType();
```

setClassname()

Sets the name of the class that implements this provider.

Syntax

```
public void setClassname( String classname);
```

Parameter	Description
classname	The name of the implementing class.

setId()

Sets the provider id.

Syntax

```
public void setId( String id);
```

Parameter	Description
id	The unique provider id.

setOptions()

Sets the options.

Syntax

```
public void setOptions( Hashtable options);
```

Parameter	Description
options	The name-value pairs that represent the provider implementation-specific options for this service.

setProviderType()

Sets the provider type.

Syntax

```
public void setProviderType( String providerType);
```

Parameter	Description
providerType	The provider type.

toString()

Writes out the service deployment descriptor to String.

Syntax

```
public String toString();
```

toXML()

Writes out the service deployment descriptor as XML.

Syntax

```
public void toXML( Writer pr);
```

Parameter	Description
pr	The writer for the XML output.

RequestContext Class

RequestContext defines all of the context for a SOAP request, including information that is passed to the provider and information that the provider must set before returning. The provider is given the request Envelope and is therefore responsible to unmarshall the request parameters. Similarly, the provider is required to marshall the response, although the response envelope must also be set by the provider, as it may be needed by a pluggable handler. The following information is provided by the SOAP engine to the Provider, meaning that the provider can utilize this information in `Provider.invoke()`:

- `getEnvelope` - the envelope containing the request
- `getServiceDeploymentDescriptor` - the service deployment descriptor for the service in which the method is being invoked
- `getServiceId` - the URI of the service
- `getUserContext` - the security context describing the user invoking the method in the service
- `getMethodName` - the name of the method being invoked in the service.

The following information must be given by the Provider to the SOAP engine:

- `setResponseBytes` - this is the marshalled response. Given a Response, it can be created by building the response envelope and then marshalling the envelope.
- `setResponseEnvelope` - this is the response envelope, which is logically equivalent to the response bytes.
- `getRequestEncodingStyle` - the encoding style to use for the response in case of an error (if not set, defaults to `Constants.NS_URI_SOAP_ENC`, which is SOAP encoding). If the provider cares about this, it should set this value as soon as it can in case of an exception. The provider might use the same encoding as the request or as one of the parameters.

Syntax

```
public class RequestContext extends Object
```

Table 12–12 Summary of Methods of RequestContext

Method	Description
RequestContext() on page 12-32	Default constructor for this class.
getMethodName() on page 12-32	Returns the method name being invoked for this SOAP request.
getRequestEncodingStyle() on page 12-32	Returns the encoding style that was used on the request.
getRequestEnvelope() on page 12-32	Returns the envelope that represents the actual SOAP request.
getResponseBytes() on page 12-32	Returns the response stream for this SOAP request.
getResponseEnvelope() on page 12-32	Returns the envelope that represents the SOAP response.
getResponseMap() on page 12-33	Returns the mapping registry that must be used to serialize the SOAP response.
getServiceDeploymentDescriptor() on page 12-33	Returns the service deployment descriptor for the requested service.
getServiceId() on page 12-33	Returns the service id (URI) for this SOAP request.
getUserContext() on page 12-33	Returns the user context for this SOAP request.
setMethodName() on page 12-33	Sets the method name for this SOAP request.
setRequestEncodingStyle() on page 12-33	Sets the encoding style that was used on the request.
setRequestEnvelope() on page 12-34	Sets the envelope that represents the actual SOAP request.
setResponseBytes() on page 12-34	Sets the response stream for this SOAP request.
setResponseEnvelope() on page 12-34	Sets the envelope that represents the SOAP response.
setResponseMap() on page 12-35	Sets the mapping registry that must be used to serialize the SOAP response envelope.
setServiceDeploymentDescriptor on page 12-35	Sets the service deployment descriptor for the requested service.
setServiceId() on page 12-35	Sets the service id (URI) for this SOAP request.
setUserContext() on page 12-35	Sets the user context for this SOAP request.

RequestContext()

Default constructor for this class.

Syntax

```
public RequestContext();
```

getMethodName()

Returns the method name being invoked for this SOAP request.

Syntax

```
public String getMethodName();
```

getRequestEncodingStyle()

Returns the encoding style that was used on the request.

Syntax

```
public String getRequestEncodingStyle();
```

getRequestEnvelope()

Returns the envelope that represents the actual SOAP request.

Syntax

```
public Envelope getRequestEnvelope();
```

getResponseBytes()

Returns the response stream for this SOAP request.

Syntax

```
public ByteArrayOutputStream getResponseBytes();
```

getResponseEnvelope()

Returns the envelope that represents the SOAP response.

Syntax

```
public Envelope getResponseEnvelope();
```

Parameter	Description
smr	The mapping registry for the SOAP response envelope.

getResponseMap()

Returns the mapping registry that must be used to serialize the SOAP response.

Syntax

```
public SOAPMappingRegistry getResponseMap();
```

getServiceDeploymentDescriptor()

Returns the service deployment descriptor for the requested service, or NULL if the provider is an AutonomousProvider.

Syntax

```
public ServiceDeploymentDescriptor getServiceDeploymentDescriptor();
```

getServiceId()

Returns the service id (URI) for this SOAP request.

Syntax

```
public String getServiceId();
```

getUserContext()

Returns the user context for this SOAP request.

Syntax

```
public UserContext getUserContext();
```

setMethodName()

Sets the method name for this SOAP request. The method name is in the envelope, but it can be "cached" here by the server as a convenience.

Syntax

```
public void setMethodName( String methodName);
```

Parameter	Description
methodName	The method name that is being invoked in the service.

setRequestEncodingStyle()

Sets the encoding style that was used on the request.

Syntax

```
public void setRequestEncodingStyle( String requestEncodingStyle);
```

Parameter	Description
requestEncodingStyle	The request encoding style.

setRequestEnvelope()

Sets the envelope that represents the actual SOAP request.

Syntax

```
public void setRequestEnvelope( Envelope envelope);
```

Parameter	Description
envelope	The SOAP envelope.

setResponseBytes()

Sets the response stream for this SOAP request.

Syntax

```
public void setResponseBytes( ByteArrayOutputStream bytes);
```

Parameter	Description
bytes	The ByteArrayOutputStream that contains the response.

setResponseEnvelope()

Sets the envelope that represents the SOAP response.

Syntax

```
public void setResponseEnvelope( Envelope envelope);
```

Parameter	Description
envelope	The SOAP response envelope.

setResponseMap()

Sets the mapping registry that must be used to serialize the SOAP response envelope.

Syntax

```
public void setResponseMap( SOAPMappingRegistry smr);
```

setServiceDeploymentDescriptor

Sets the service deployment descriptor for the requested service.

Syntax

```
public void setServiceDeploymentDescriptor(
                  ServiceDeploymentDescriptor serviceDeploymentDescriptor);
```

Parameter	Description
serviceDeploymentDescriptor	The service deployment descriptor for this request.

setServiceId()

Sets the service id (URI) for this SOAP request.

Syntax

```
public void setServiceId( String serviceId);
```

Parameter	Description
serviceId	The URI for the service to which this request is directed.

setUserContext()

Sets the user context for this SOAP request.

Syntax

```
public void setUserContext( UserContext userContext);
```

Parameter	Description
userContext	The user context.

ServiceDeploymentDescriptor Class

`ServiceDeploymentDescriptor` defines the deployment information for a SOAP service, independent of its provider type. The class supports any number of named provider options, which allows the descriptor to be easily extended (without code changes) for new types of providers.

Syntax

```
public final class ServiceDeploymentDescriptor extends Object implements
Serializable
```

Table 12–13 Fields of ServiceDeploymentDescriptor

Field	Syntax	Description
`SERVICE_TYPE_RPC`	`public static final int SERVICE_TYPE_RPC`	Indicates the service is RPC based.
`SERVICE_TYPE_MESSAGE`	`public static final int SERVICE_TYPE_MESSAGE`	Indicates the service is message based.
`SCOPE_REQUEST`	`public static final int SCOPE_REQUEST`	Indicates that a fresh service instance should be allocated for each request.
`SCOPE_SESSION`	`public static final int SCOPE_SESSION`	Indicates that all requests within the same session will be served by the same service instance.
`SCOPE_APPLICATION`	`public static final int SCOPE_APPLICATION`	Indicates that all requests will be served by the same service instance.

Table 12–14 Summary of Methods of ServiceDeploymentDescriptor

Method	Description
ServiceDeploymentDescriptor() on page 12-39	Constructs a new service descriptor.
buildFaultRouter() on page 12-39	Returns the fault router that is built from the service's fault listeners.
buildSOAPMappingRegistry() on page 12-39	Generates an XML serialization registry from all the type mappings registered into a deployment descriptor.

Table 12–14 (Cont.) Summary of Methods of ServiceDeploymentDescriptor

Method	Description
buildSqlClassMap() on page 12-40	Generates a map from SQL type to Java Class using the type mapping information from the deployment descriptor. Throws SOAPException if failed to generate map.
fromXML() on page 12-40	Populates the ServiceDeploymentDescriptor with information from the given document, which is the XML representation of the descriptor.
getDefaultSMRClass() on page 12-40	Returns the default SOAP mapping registry class.
getFaultListener() on page 12-41	Returns list of class names that are fault listeners for this service.
getId() on page 12-41	Returns the service id, which is a URI.
getMethods() on page 12-41	Returns the list of methods that are provided by this service.
getProviderId() on page 12-41	Returns the provider id for this service.
getProviderOptions() on page 12-41	Returns the name-value pairs that represent the provider-specific options for this service.
getProviderType() on page 12-41	Returns the provider type.
getScope() on page 12-41	Returns the scope, which is one of the SCOPE_xxx constants.
getServiceType() on page 42	Returns the service type, which is one of the SERVICE_TYPE_xxx constants.
getSqlMap() on page 12-42	Returns the SQL type to Java type map.
getTypeMappings() on page 42	Returns the XML-Java type mappings, which define how to deserialize XML into Java and serialize Java into XML.
isMethodValid() on page 12-42	Determines if the given method is valid for this service.
setDefaultSMRClass() on page 12-42	Sets the default SOAP mapping registry class.
setFaultListener() on page 12-43	Sets the fault listener list.
setId() on page 12-43	Sets the service id, which must be a valid URI.
setMethods() on page 12-43	Sets the list of methods that are provided by this service.

Table 12–14 (Cont.) Summary of Methods of ServiceDeploymentDescriptor

Method	Description
setProviderId() on page 12-43	Sets the id of the provider for this service.
setProviderOptions() on page 12-44	Sets the provider-specific options.
setProviderType() on page 12-44	Sets the provider type.
setScope() on page 12-44	Sets the execution scope.
setServiceType() on page 12-45	Sets the service type.
setSqlMap() on page 12-45	Sets the map that maps from SQL type to Java type.
setTypeMappings() on page 12-45	Sets the XML-Java type mappings, which define how to deserialize XML into Java and serialize Java into XML.
toXML() on page 12-46	Writes out the service deployment descriptor as XML.
toString() on page 12-46	Returns a printable representation of this descriptor.

ServiceDeploymentDescriptor()

Constructs a new service descriptor.

Syntax

```
public ServiceDeploymentDescriptor();
```

buildFaultRouter()

Returns the fault router that is built from the service's fault listeners.

Syntax

```
public SOAPFaultRouter buildFaultRouter();
```

buildSOAPMappingRegistry()

Generates an XML serialization registry from all the type mappings registered into a deployment descriptor.

Syntax

```
public static SOAPMappingRegistry buildSOAPMappingRegistry(
                                        ServiceDeploymentDescriptor sdd);
```

Parameter	Description
sdd	The service deployment descriptor.

buildSqlClassMap()

Generates a map from SQL type to Java Class using the type mapping information from the deployment descriptor. Throws SOAPException if failed to generate map.

Syntax

```
public static Hashtable buildSqlClassMap( ServiceDeploymentDescriptor sdd);
```

Parameter	Description
sdd	The service deployment descriptor to use.

fromXML()

Populates the ServiceDeploymentDescriptor with information from the given document, which is the XML representation of the descriptor; returns this ServiceDeploymentDescriptor. Throws `IllegalArgumentException` if invalid document.

Syntax

```
public static ServiceDeploymentDescriptor fromXML( Element root);
```

Parameter	Description
root	The root of the XML document that represents the service descriptor.

getDefaultSMRClass()

Returns the default SOAP mapping registry class.

Syntax

```
public String getDefaultSMRClass();
```

getFaultListener()

Returns list of class names that are fault listeners for this service.

Syntax

```
public String[] getFaultListener();
```

getId()

Returns the service id, which is a URI.

Syntax

```
public String getId();
```

getMethods()

Returns the list of methods that are provided by this service.

Syntax

```
public String[] getMethods();
```

getProviderId()

Returns the provider id for this service.

Syntax

```
public String getProviderId();
```

getProviderOptions()

Returns the name-value pairs that represent the provider-specific options for this service.

Syntax

```
public Hashtable getProviderOptions();
```

getProviderType()

Returns the provider type.

Syntax

```
public String getProviderType();
```

getScope()

Returns the scope, which is one of the `SCOPE_xxx` constants.

Syntax

```
public int getScope();
```

getServiceType()

Returns the service type, which is one of the SERVICE_TYPE_xxx constants.

Syntax

```
public int getServiceType();
```

getSqlMap()

Returns the SQL type to Java type map.

Syntax

```
public Hashtable getSqlMap();
```

getTypeMappings()

Returns the XML-Java type mappings, which define how to deserialize XML into Java and serialize Java into XML.

Syntax

```
public TypeMapping[] getTypeMappings();
```

isMethodValid()

Determines if the given method is valid for this service. Returns `TRUE` if the method is valid for this service, `FALSE` otherwise.

Syntax

```
public boolean isMethodValid( String methodName);
```

setDefaultSMRClass()

Sets the default SOAP mapping registry class.

Syntax

```
public void setDefaultSMRClass( String defaultSMRClass);
```

Parameter	Description
`defaultSMRClass`	The default SOAP mapping registry class.

setFaultListener()

Sets the fault listener list.

Syntax

```
public void setFaultListener( String faultListener[]);
```

Parameter	Description
faultListener	The list of class names that are fault listeners for this service.

setId()

Sets the service id, which must be a valid URI.

Syntax

```
public void setId( String id);
```

Parameter	Description
id	The service URI.

setMethods()

Sets the list of methods that are provided by this service.

Syntax

```
public void setMethods( String methods[]);
```

Parameter	Description
methods	The list of provided methods.

setProviderId()

Sets the id of the provider for this service.

Syntax

```
public void setProviderId( String providerId);
```

Parameter	Description
providerId	The provider's id for this service.

setProviderOptions()

Sets the provider-specific options.

Syntax

```
public void setProviderOptions( Hashtable providerOptions);
```

Parameter	Description
providerOptions	The name-value pairs that represent provider-specific options.

setProviderType()

Sets the provider type.

Syntax

```
public void setProviderType( String providerType);
```

Parameter	Description
providerType	The provider type, which can be any string. The provider type is used to validate the XML service descriptor (for the provider-specific options).

setScope()

Sets the execution scope.

Syntax

```
public void setScope( int scope);
```

Parameter	Description
scope	The execution scope, which is one of the SCOPE_xxx constants.

setServiceType()

Sets the service type.

Syntax

```
public void setServiceType( int serviceType);
```

Parameter	Description
serviceType	The service type, which is one of the SERVICE_TYPE_xxx constants.

setSqlMap()

Sets the map that maps from SQL type to Java type.

Syntax

```
public void setSqlMap( Hashtable sqlMap);
```

Parameter	Description
sqlMap	The SQL type to Java class map.

setTypeMappings()

Sets the XML-Java type mappings, which define how to deserialize XML into Java and serialize Java into XML.

Syntax

```
public void setTypeMappings( TypeMapping typeMappings[]);
```

Parameter	Description
typeMappings	The type mappings.

toXML()

Writes out the service deployment descriptor as XML.

Syntax

```
public void toXML( Writer pr);
```

Parameter	Description
pr	The writer for the XML output.

toString()

Returns a printable representation of this descriptor.

Syntax

```
public String toString();
```

SOAPServerContext Class

SOAPServerContext defines the context of the SOAP server that is independent of the type of container in which the server is running.

Syntax

```
public class SOAPServerContext extends Object
```

Table 12–15 Summary of Methods of SOAPServerContext

Method	Description
SOAPServerContext() on page 12-47	Default constructor.
getAttribute() on page 12-48	Returns the attribute with the given name, or null if there is no attribute by that name.
getAttributeNames() on page 12-48	Returns an Enumeration containing the attribute names available within this SOAP context.
getGlobalContext() on page 12-48	Returns the global context.
getLogger() on page 12-48	Returns the SOAP logger.
removeAttribute() on page 12-48	Removes the attribute with the given name from the context.
setAttribute() on page 12-49	Binds an object to a given attribute name in this SOAP context.
setGlobalContext () on page 12-49	Set the global context, which contains SOAP server-wide objects.
setLogger() on page 12-49	Set the logger, which is used for text-based logging of informational and debug messages.

SOAPServerContext()

Default constructor.

Syntax

```
public SOAPServerContext();
```

getAttribute()

Returns an Object containing the value of the attribute. or NULL if there is no attribute by that name.

Syntax

```
public Object getAttribute( String name);
```

Parameter	Description
name	A String specifying the name of the attribute to get.

getAttributeNames()

Returns an Enumeration containing the attribute names available within this SOAP context.

Syntax

```
public Enumeration getAttributeNames();
```

getGlobalContext()

Returns the global context that contains SOAP server-wide objects, or null if the attribute is not set.

Syntax

```
public Hashtable getGlobalContext();
```

getLogger()

Returns the SOAP logger, which is used to log informational and debug messages.

Syntax

```
public Logger getLogger();
```

removeAttribute()

Removes the attribute with the given name from the context. After removal, subsequent calls to getAttribute(java.lang.String) to retrieve the attribute's value will return null.

Syntax

```
public void removeAttribute( String name);
```

Parameter	Description
name	A String specifying the name of the attribute to be removed.

setAttribute()

Binds an object to a given attribute name in this SOAP context. If the name specified is already used for an attribute, this method will remove the old attribute and bind the name to the new attribute. Neither the name nor the object may be `NULL`.

Syntax

```
public void setAttribute( String name,
                          Object object);
```

Parameter	Description
`name`	A non-null String specifying the name of the attribute.
`object`	A non-null Object representing the attribute to be bound.

setGlobalContext ()

Sets the global context, which contains SOAP server-wide objects.

Syntax

```
public void setGlobalContext( Hashtable globalContext);
```

Parameter	Description
`globalContext`	The global context.

setLogger()

Sets the logger, which is used for text-based logging of informational and debug messages.

Syntax

```
public void setLogger( Logger logger);
```

Parameter	Description
`logger`	The SOAP logger.

UserContext Class

UserContext defines the user context for a SOAP service request. Several attributes are pre-defined, and set and get methods are provided for those. In addition, the provider may define additional attributes using getAttribute and setAttribute. The `HttpServlet` and `HttpSession` do not really belong here, but they are required by the JavaProvider.

Syntax

```
public class UserContext extends Object
```

Table 12–16 Summary of Methods of UserContext

Method	Description
UserContext() on page 12-52	Default constructor.
getAttribute() on page 12-52	Returns the attribute with the given name
getAttributeNames() on page 12-52	Returns an Enumeration containing the attribute names available within this SOAP context.
getCertificate() on page 53	Returns the user certificate for the user making SOAP request.
getHttpServlet() on page 12-53	Returns the HttpServlet that is processing the SOAP request.
getHttpSession() on page 12-53	Returns the HTTP session for the SOAP request
getRemoteAddress() on page 12-53	Returns the Internet Protocol (IP) address of the remote client that sent the request.
getRemoteHost() on page 12-53	Returns the host name of the remote client that sent the request.
getRequestURI() on page 12-53	Returns the URI of the request.
getSecureChannel() on page 54	Returns an indication whether the channel is secure.
getUsername() on page 54	Returns the protocol-specific username for the SOAP request.
removeAttribute() on page 12-54	Removes the attribute with the given name from the context.
setAttribute() on page 12-54	Binds an object to a given attribute name in this SOAP context.

Table 12–16 (Cont.) Summary of Methods of UserContext

Method	Description
setCertificate() on page 12-55	Sets the user certificate.
setHttpServlet() on page 12-55	Sets the HTTP servlet.
setHttpSession() on page 55	Sets the HTTP session.
setRemoteAddress() on page 12-56	Sets the remote IP address of the client.
setRemoteHost() on page 12-56	Sets the host name of the remote client making the SOAP request.
setRequestURI() on page 12-56	Sets the URI of the request.
setSecureChannel() on page 12-56	Sets the indicator of whether the channel is secure.
setUsername() on page 12-57	Sets the protocol-specific username.

UserContext()

Default constructor.

Syntax

```
public UserContext();
```

getAttribute()

Returns the attribute with the given name, or `NULL` if there is no attribute by that name.

Syntax

```
public Object getAttribute( String name);
```

Parameter	Description
`name`	A String specifying the name of the attribute

getAttributeNames()

Returns an `Enumeration` containing the attribute names available within this SOAP context.

Syntax

```
public Enumeration getAttributeNames();
```

getCertificate()

Returns the user certificate for the user making SOAP request, or null if this attribute is not set.

Syntax

```
public Object getCertificate();
```

getHttpServlet()

Returns the HttpServlet that is processing the SOAP request, or null if the servlet attribute is not set.

Syntax

```
public HttpServlet getHttpServlet();
```

getHttpSession()

Returns the HTTP session for the SOAP request, or null if the session attribute is not set.

Syntax

```
getHttpSession public HttpSession getHttpSession();
```

getRemoteAddress()

Returns the Internet Protocol (IP) address of the remote client that sent the request.

Syntax

```
public String getRemoteAddress();
```

getRemoteHost()

Returns the host name of the remote client that sent the request.

Syntax

```
public String getRemoteHost();
```

getRequestURI()

Returns the URI of the request.

Syntax

```
public String getRequestURI();
```

getSecureChannel()

Returns an indication whether the channel is secure; TRUE if the channel is secure, FALSE otherwise.

Syntax

```
public boolean getSecureChannel();
```

getUsername()

Returns the protocol-specific username for the SOAP request, or null if this attribute is not set.

Syntax

```
public String getUsername();
```

removeAttribute()

Removes the attribute with the given name from the context. After removal, subsequent calls to getAttribute(java.lang.String) to retrieve the attribute's value will return NULL.

Syntax

```
public void removeAttribute( String name);
```

Parameter	Description
name	A non-null String specifying the name of the attribute.

setAttribute()

Binds an object to a given attribute name in this SOAP context. If the name specified is already used for an attribute, this method will remove the old attribute and bind the name to the new attribute. Neither the name nor the object may be NULL.

Syntax

```
public void setAttribute( String name,
                          Object object);
```

Parameter	Description
name	A non-null String specifying the name of the attribute.

Parameter	Description
object	An non-null Object representing the attribute to be bound.

setCertificate()

Sets the user certificate.

Syntax

```
public void setCertificate( Object certificate);
```

Parameter	Description
certificate	The user certificate for the user making the SOAP request.

setHttpServlet()

Sets the HTTP servlet.

Syntax

```
public void setHttpServlet( HttpServlet servlet);
```

Parameter	Description
servlet	The HttpServlet that is processing the SOAP request.

setHttpSession()

Sets the HTTP session.

Syntax

```
public void setHttpSession( HttpSession session);
```

Parameter	Description
servlet	The HttpSession for the SOAP request.

setRemoteAddress()

Sets the remote IP address of the client.

Syntax

```
public void setRemoteAddress( String remoteAddress);
```

Parameter	Description
remoteAddress	The IP address of the client making the SOAP request.

setRemoteHost()

Sets the host name of the remote client making the SOAP request.

Syntax

```
public void setRemoteHost( String remoteHost);
```

Parameter	Description
remoteHost	The host name of the client making the SOAP request.

setRequestURI()

Sets the URI of the request.

Syntax

```
public void setRequestURI( String uri);
```

Parameter	Description
uri	Request URI.

setSecureChannel()

Sets the indicator of whether the channel is secure.

Syntax

```
public void setSecureChannel( boolean secureChannel);
```

Parameter	Description
secureChannel	TRUE if the channel is secure, FALSE otherwise.

setUsername()

Sets the protocol-specific username.

Syntax

```
public void setUsername( String username);
```

Parameter	Description
username	The protocol-specific username for the SOAP request.

oracle.soap.transport Package

Contains the OracleSOAPTransport Interface and provides support for Oracle SOAP in the XDK for Java.

OracleSOAPTransport Interface

This interface defines Oracle specific transport extensions.

Syntax

```
public interface OracleSOAPTransport extends SOAPTransport
```

Table 12–17 Summary of Methods of OracleSOAPTransport

Method	Description
close() on page 12-59	Close the transport and perform any clean up.
getProperties() on page 12-59	Returns the connection properties.
setProperties() on page 12-59	Sets the connection properties.

close()

Closes the transport and performs clean up.

Syntax

```
public abstract void close();
```

getProperties()

Returns the connection properties.

Syntax

```
public abstract Properties getProperties();
```

setProperties()

Sets the connection properties.

Syntax

```
public abstract void setProperties( Properties prop);
```

Parameter	Description
prop	Connection properties.

oracle.soap.transport.http Package

Package oracle.soap.transport.http contains OracleSOAPHTTPConnection Class, which implements `OracleSOAPTransport`. The Oracle SOAP client API supports a pluggable transport, allowing the client to easily change the transport. Available transports include HTTP and HTTPS (secure HTTP).

OracleSOAPHTTPConnection Class

This class implements `OracleSOAPTransport`.

Syntax

```
public class OracleSOAPHTTPConnection extends Object
```

Table 12–18 Fields of OracleSOAPHTTPConnection

Field	Syntax	Description
`ALLOW_USER_INTERACTION`	`public static final String ALLOW_USER_INTERACTION`	Property to set user interaction.
`AUTH_TYPE`	`public static final String AUTH_TYPE`	Property used for defining http auth type, (basic/digest.
`CIPHERS`	`public static final String CIPHERS`	Property used for defining cipher suites used for HTTPS, a colon separated list of cipher suites.
`PASSWORD`	`public static final String PASSWORD`	Property used for defining http password.
`PROXY_AUTH_TYPE`	`public static final String PROXY_AUTH_TYPE`	Property used for defining proxy auth type, basic/digest.
`PROXY_HOST`	`public static final String PROXY_HOST`	Property used for defining proxy host.
`PROXY_PASSWORD`	`public static final String PROXY_PASSWORD`	Property used for defining proxy password.
`PROXY_PORT`	`public static final String PROXY_PORT`	Property used for defining proxy port.
`PROXY_USERNAME`	`public static final String PROXY_USERNAME`	Property used for defining proxy username.
`STATUS_LINE`	`public static final String STATUS_LINE`	Property used to get HTTP status line from HTTP headers, `getHeaders()`.
`USERNAME`	`public static final String USERNAME`	Property used for defining http username.

Table 12–18 (Cont.) Fields of OracleSOAPHTTPConnection

Field	Syntax	Description
WALLET_LOCATION	public static final String WALLET_LOCATION	Property used for defining wallet location used for HTTPS.
WALLET_PASSWORD	public static final String WALLET_PASSWORD	Property used for defining wallet password used for HTTPS.

Table 12–19 Summary of Methods of OracleSOAPHTTPConnection

Member	Description
OracleSOAPHTTPConnection() on page 12-62	Constructs a new instance of `OracleSOAPHTTPConnection` from given properties.
close() on page 12-63	Closes the connection.
finalize() on page 12-63	Finalizes the connection.
getHeaders() on page 12-63	Returns a hashtable containing all the headers to headers generated by the protocol.
getProperties() on page 12-63	Returns the connection properties.
receive() on page 63	Returns a buffered reader from which the received response is read.
send() on page 12-63	Requests that an envelope be posted to the given URL.
setProperties() on page 12-64	Sets the connection properties.

OracleSOAPHTTPConnection()

Constructs a new instance of `OracleSOAPHTTPConnection` from given properties.

Syntax

```
public OracleSOAPHTTPConnection( Properties prop);
```

Parameter	Description
prop	Connection properties.

close()

Closes the connection. Once this method has been called, the `BufferedReader` returned by `receive` method may be closed and should not be used. Calling this method will free resources without having the garbage collector run.

Syntax

```
public void close();
```

finalize()

Finalizes the connection.

Syntax

```
public void finalize();
```

getHeaders()

Returns a hashtable containing all the headers to headers generated by the protocol. SOAP clients should not use this method directly but use `org.apache.soap.rpc.Call()` instead.

Syntax

```
public Hashtable getHeaders();
```

getProperties()

Returns the connection properties.

Syntax

```
public Properties getProperties();
```

receive()

Returns a buffered reader from which the received response is read, or `null` if the response is not received. SOAP clients should not use this method directly but use `org.apache.soap.rpc.Call()` instead.

Syntax

```
public BufferedReader receive();
```

send()

Requests that an envelope be posted to the given URL. The response (if any) is retrieved by calling the `receive()` function. SOAP clients should not use this

method directly, but should instead use `org.apache.soap.rpc.Call()`. Throws `SOAPException` with appropriate reason code if there are errors.

Syntax

```
public void send( URL sendTo,
                  String action,
                  Hashtable headers,
                  Envelope env,
                  SOAPMappingRegistry smr,
                  int timeout);
```

Parameter	Description
`sendTo`	The URL to which the envelope is sent.
`action`	The SOAPAction header field value.
`headers`	Any other header fields to go to as protocol headers.
`env`	The envelope to send.
`smr`	The XML<->Java type mapping registry, passed on.
`ctx`	The request SOAPContext.

setProperties()

Sets the connection properties.

Syntax

```
public void setProperties( Properties prop);
```

Parameter	Description
`prop`	Connection properties.

oracle.soap.util.xml Package

Package oracle.soap.util.xml contains the XmlUtils Class.

XmlUtils Class

The XmlUtils class implements Oracle- specific transport extensions in OracleSOAPTransport. The APIs of this class enable SOAP clients to generate the XML documents that compose a request for a SOAP service, and to handle the SOAP response. Oracle SOAP processes requests from any client that sends a valid SOAP request.

Syntax

```
public class XmlUtils
```

Table 12–20 Summary of Methods of XmlUtils

Member	Description
XmlUtils() on page 12-66	Default constructor.
extractServiceId() on page 12-66	Returns the service id from the envelope.
extractMethodName() on page 12-67	Returns the method name from the envelope.
parseXml() on page 12-67	Parses the given XML file and returns the XML document.
createDocument() on page 12-68	Creates a Document.

XmlUtils()

Default constructor.

Syntax

```
public XmlUtils();
```

extractServiceId()

Returns the service id from the envelope. It is the namespace URI of the first body entry. Throws `SOAPException` if unable to get service URI from envelope.

Syntax

```
public static String extractServiceId(Envelope envelope);
```

Parameter	Description
envelope	The SOAP envelope.

extractMethodName()

Returns the method name from the envelope. It is the name of the first body entry. Throws SOAPException if unable to get method name from envelope.

Syntax

```
public static String extractMethodName( Envelope envelope);
```

Parameter	Description
envelope	The SOAP envelope.

parseXml()

Parses the given XML file and returns the XML document. Throws SOAPException if file not found, there is a parse error, or I/O errors. The options are described in the following table.

Syntax	Description
public static Document parseXml(String filename);	Parses the given XML file and returns the XML document, given the filename.
public static Document parseXml(Reader reader);	Parses the given XML file and returns the XML document from a reader.
public static Document parseXml(InputStream is);	Parses the given XML file and returns the XML document from an input stream.

Parameter	Description
filename	The full path to the XML file.
reader	Reader for XML.
is	The input stream source.

createDocument()

Creates a `Document`. Throws a `SOAPException` if cannot create `Document`.

Syntax

```
public static Document createDocument();
```

Part II

C Support for XML

This section contains the following chapters:

- Chapter 13, "XML Schema Processor for C"
- Chapter 14, "XML Parser for C"

13

XML Schema Processor for C

The schema API is very simple: initialize, validate,...validate, terminate.

The validation process is go/no-go. Either the document is valid with respect to the schemas or it is invalid. When it is valid, a zero error code is returned. When it is invalid, a non-zero error code is returned indicating the problem. There is no distinction between warnings and errors; all problems are errors and considered fatal: validation stops immediately.

As schemas are encountered, they are loaded and preserved in the schema context. No schema is loaded more than once during a session. There is no clean up call similar to `xmlclean`. Hence, if you need to release all memory and reset state before validating a new document, you must terminate the context and start over.

This chapter contains the following sections:

- XML Schema Methods for C

See Also:

- *Oracle Application Developer's Guide - XML*

XML Schema Methods for C

Table 13–1 summarizes the methods of the C parser.

Table 13–1 Summary of Methods of XML Schema for C

Method	Description
schemaInitialize() on page 13-2	Initializes the XML schema processor.
schemaValidate() on page 13-2	Validate an instance document against a schema
schemaTerminate() on page 13-3	Terminates (tears down) the schema processor

schemaInitialize()

Initializes the XML schema processor. Must be called before the processor can be used to validate any documents. The XML parser context is used to allocate memory for the schema context. The schema context is returned, and must be passed to all subsequent schema functions. This context pointer is opaque-- you cannot reference its members. If the return context is NULL, initialization failed and err will be set with the numeric error code indicating the problem.

Syntax

```
xsdctx *schemaInitialize(xmlctx *ctx, uword *err)
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context
err	OUT	Returned error code

schemaValidate()

Validates an instance document against a schema or schemas.The schema context returned by `schemaInitialize` must be passed in. The document to be validated is specified by the XML parser context `inst` used to parse the document. Note the document must already have been parsed. If no schemas are explicitly referenced in the instance document, the default schema (specified by URL) is assumed. If the document *does* specify all necessary schemas, and a default schema is also supplied,

the default will be ignored. If the document does not reference any schemas and no default is supplied, an error will result.

Syntax

```
uword schemaValidate(xsdctx *scctx, xmlctx *inst, oratext *schema)
```

Parameter	IN/OUT	Description
scctx	IN	Schema context
inst	IN	Instance document context
schema	IN	URL of default schema

Comments

schemaTerminate()

Terminates (shuts down) the schema processor, freeing all memory allocated on the original XML parser context passed to `schemaInitialize`. After termination, the schema context is no longer valid. To continue using the schema processor, a new schema must be created with `schemaInitialize`.

Syntax

```
void schemaTerminate(xsdctx *scctx)
```

Parameter	IN/OUT	Description
scctx	IN	Schema context

14

XML Parser for C

This chapter describes the C language implementation of XML Parser.

This chapter contains the following sections:

- Parser APIs
- XSLT API
- W3C SAX APIs
- W3C DOM APIs
- Namespace APIs
- Datatypes

See Also:

- *Oracle Application Developer's Guide - XML*

Parser APIs

Extensible Markup Language (XML) describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure. A software module called an XML processor is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the application.

This C implementation of the XML processor (or parser) followed the W3C XML specification (rev REC-xml-19980210) and included the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

The following is the default behavior of this parser:

- The character set encoding is UTF-8. If all your documents are ASCII, you are encouraged to set the encoding to US-ASCII for better performance.
- Messages are printed to stderr unless msghdlr is given.
- A parse tree which can be accessed by DOM APIs is built unless saxcb is set to use the SAX callback APIs. Note that any of the SAX callback functions can be set to NULL if not needed.
- The default behavior for the parser is to check that the input is well-formed but not to check whether it is valid. The flag XML_FLAG_VALIDATE can be set to validate the input. The default behavior for whitespace processing is to be fully conformant to the XML 1.0 spec, i.e. all whitespace is reported back to the application but it is indicated which whitespace is ignorable. However, some applications may prefer to set the XML_FLAG_DISCARD_WHITESPACE which will discard all whitespace between an end-element tag and the following start-element tag.

Calling Sequence

Parsing a single document:

```
xmlinit, xmlparsexxx, xmlterm
```

```
Parsing multiple documents, but only the latest document needs to be available:
```

```
xmlinit, xmlparsexxx, xmlclean, xmlparsexxx, xmlclean ... xmlterm
```

Parsing multiple documents, all document data must be available:

```
xmlinit, xmlparsexxx, xmlparsexxx ... xmlterm
```

Memory

The memory callback functions specified in `memcb` may be used if you wish to use your own memory allocation. If they are used, all of the functions should be specified.

The memory allocated for Parameters passed to the SAX callbacks or for nodes and data stored with the DOM parse tree will not be freed until one of the following is done:

- `xmlparsexxx` is called to parse another document.
- `xmlclean` is called.
- `xmlterm` is called.

Thread Safety

If threads are forked off somewhere in the midst of the init-parse-terminate sequence of calls, you will get unpredictable behavior and results.

Table 14–1 summarizes the methods of the C parser.

Table 14–1 Summary of the Methods of the C Parser

Method	Description
xmlinit() on page 14-4	Initialize XML parser
xmlclean() on page 14-5	Clean up memory used during parse
xmlparse() on page 14-6	Parse a document specified by a URL
xmlparsebuf() on page 14-7	Parse a document that's resident in memory
xmlparsefile() on page 14-7	Parse a document from the filesystem

Table 14–1 (Cont.) Summary of the Methods of the C Parser

Method	Description
xmlparsestream() on page 14-8	Parse a document from a user-defined stream
xmlterm() on page 14-8	Shut down XML parser
createDocument() on page 14-9	Create a new document
isStandalone() on page 14-9	Return document's standalone flag
isSingleChar() on page 14-9	Return single/multibyte encoding flag
getEncoding() on page 14-10	Return name of document's encoding

xmlinit()

Initializes the XML parser. It must be called before any parsing can take place.

The C version of this call Returns the XML context on success, and sets the user's `err` argument on error. As usual, a zero error code means success, non-zero indicates a problem.

This function should only be called once before starting the processing of one or more XML files. xmlterm() should be called after all processing of XML files has completed.

Error codes are: `XMLERR_LEH_INIT, XMLERR_BAD_ENCODING, XMLERR_NLS_INIT, XMLERR_NO_MEMORY, XMLERR_NULL_PTR`.

For C, all arguments may be NULL except for `err`. For C++, all arguments have default values and may be omitted if not needed.

By default, the character set encoding is UTF-8. If all your documents are ASCII, you are encouraged to set the encoding to US-ASCII for better performance.

By default, messages are printed to `stderr` unless `msghdlr` is given.

By default, a parse tree is built (accessible by DOM APIs) unless `saxcb` is set (in which case the SAX callback APIs are invoked). Note that any of the SAX callback functions can be set to NULL if not needed.

The memory callback functions memcb may be used if you wish to use your own memory allocation. If they are used, all of the functions should be specified.

The Parameters `msgctx`, `saxcbctx`, and `memcbctx` are structures that you may define and use to pass information to your callback routines for the message handler, SAX functions, or memory functions, respectively. They should be set to

NULL if your callback functions do not need any additional information passed in to them.

The `lang` parameter is not used currently and may be set to NULL. It will be used in future releases to determine the language of the error messages.

Syntax

```
xmlctx *xmlinit(uword *err, const oratext *encoding,
                void (*msghdlr)(void *msgctx, const oratext *msg,
                                uword errcode),
                void *msgctx, const xmlsaxcb *saxcb, void *saxcbctx,
                const xmlmemcb *memcb, void *memcbctx, const oratext *lang);
```

Parameter	IN/OUT	Description
`er`	`OUT`	The error, if any (C only).
`encoding`	`IN`	Default character set encoding
`msghdlr`	`IN`	Error message handler function
`msgctx`	`IN`	Context for the error message handler
`saxcb`	`IN`	SAX callback stgructure filled with function pointers
`saxcbctx`	`IN`	Content of SAX callbacks
`memcb`	`IN`	Memory function callbacks
`memcbctx`	`IN`	Context for memoryfunction callbacks
`lang`	`IN`	Language for error messages

xmlclean()

Frees memory used during the previous parse. This function is provided as a convenience for those who want to parse multiple documents using a single context. Before parsing the second and subsequent documents, call `xmlclean` to release memory used by the previous document.

Memory is reused internally after this call. Memory is not returned to the system until `xmlterminate`.

Syntax

```
void xmlclean(xmlctx *ctx);
```

Parameter	IN/OUT	Description
ctx	IN	The XML parser context

xmlparse()

Invokes the XML parser on an input document that is specified by a URL. The parser must have been initialized successfully with a call to xmlinit first.

Flag bits must be OR'd to override the default behavior of the parser. The following flag bits may be set:

- XML_FLAG_VALIDATE turns validation on. The default behavior is to not validate the input.
- XML_FLAG_DISCARD_WHITESPACE will discard whitespace where it appears to be insignificant. The default behavior for whitespace processing is to be fully conformant to the XML 1.0 spec, i.e. all whitespace is reported back to the application but it is indicated which whitespace is ignorable. However, some applications may prefer to set the XML_FLAG_DISCARD_WHITESPACE which will discard all whitespace between an end-element tag and the following start-element tag.
- XML_FLAG_DTD_ONLY tells the parser that the input is an external DTD only, not a complete document.
- XML_FLAG_STOP_ON_WARNING makes the parser stop immediately if any validation warnings occur. By default, validation warnings are printed but validation continues.

The memory passed to the SAX callbacks or stored with the DOM parse tree will not be freed until one of the following is done:

- xmlparse*xxx* is called to parse another document.
- xmlclean is called.
- xmlterm is called.

Syntax

```
uword xmlparse(xmlctx *ctx, const oratext *url,
               const oratext *encoding, ub4 flags);
```

Parameter	IN/OUT	Description
ctx	IN/OUT	The XML parser context
url	IN	URL of XML document
encoding	IN	*default* character set encoding
flags	IN	what options to use

xmlparsebuf()

Invokes the XML parser on a document that is resident in memory. The parser must have been initialized successfully with a call to `xmlinit` first. This function is identical to `xmlparse` except that input is taken from the user's buffer instead of from a URI, file, and so on.

Syntax

```
uword xmlparsebuf(xmlctx *ctx, const oratext *buffer, size_t len,
                  const oratext *encoding, ub4 flags);
```

Parameter	IN/OUT	Description
ctx	IN/OUT	The XML parser context
buffer	IN	pointer to document in memory
len	IN	length of the buffer
encoding	IN	default character set encoding
flags	IN	what options to use

xmlparsefile()

Invokes the XML parser on a document in the filesystem. The parser must have been initialized successfully with a call to `xmlinit` first. This function is identical to `xmlparse` except that input is taken from a file in the user's filesystem, instead of from a URL, memory buffer, and so on.

Syntax

```
uword xmlparsefile(xmlctx *ctx, const oratext *path,
                   const oratext *encoding, ub4 flags);
```

Parameter	IN/OUT	Description
ctx	IN/OUT	The XML parser context
path	IN	filesystem path of document
encoding	IN	default character set encoding
flags	IN	what options to use

xmlparsestream()

Invokes the XML parser on a document that is to be read from a user-defined stream. The parser must have been initialized successfully with a call to xmlinit first. This function is identical to xmlparse except that input is taken from a user-defined stream, instead of from a URL, file, etc. The I/O callback functions for access method XMLACCESS_STREAM must be set up first. The stream (or stream context) pointer will be available in each callback function as the ptr_xmlihdl memory of the ihdl structure. Its meaning and use are user-defined.

Syntax

```
uword xmlparsestream(xmlctx *ctx, const void *stream,
                     const oratext *encoding, ub4 flags);
```

Parameter	IN/OUT	Description
ctx	IN/OUT	The XML parser context
stream	IN	pointer to stream or stream context
encoding	IN	*default* character set encoding
flags	IN	what options to use

xmlterm()

Terminates the XML parser. It should be called after xmlinit, and before exiting the main program. This function will free all memory used by the parser and terminates the context, which may not then be reused (a new context must be created if additional parsing is to be done).

Syntax

```
uword xmlterm(xmlctx *ctx);
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context

createDocument()

Creates a new document in memory. This function is used when constructing a new document in memory. An XML document is always rooted in a node of type DOCUMENT_NODE. This function creates that root node and sets it in the context. There can be only one current document and hence only one document node; if one already exists, this function does nothing and Returns NULL.

Syntax

```
xmlnode* createDocument(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context

isStandalone()

Return value of document's *standalone* flag. This function returns the boolean value of the document's standalone flag, as specified in the <?xml?> processing instruction.

Syntax

```
boolean isStandalone(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context

isSingleChar()

Returns a flag which specifies whether the current document is encoded as single-byte characters, ASCII, or multi-byte characters, UTF-8. Compare to getEncoding, which Returns the actual name of the document's encoding.

Syntax

```
boolean isSingleChar(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context

getEncoding()

Returns the name of the current document's character encoding scheme, such as ASCII, UTF8, and so on. Compare to `isSingleChar` which just Returns a boolean flag saying whether the current encoding is single or multi-byte.

Syntax

```
oratext *getEncoding(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context

XSLT API

XSLT is a language for transforming XML documents into other XML documents.

XSLT is designed for use as part of XSL, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.

XSLT is also designed to be used independently of XSL. However, XSLT is not intended as a completely general-purpose XML transformation language. Rather it is designed primarily for the kinds of transformation that are needed when XSLT is used as part of XSL.

A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added.

A transformation expressed in XSLT is called a stylesheet. This is because, in the case when XSLT is transforming into the XSL formatting vocabulary, the transformation functions as a stylesheet.

A stylesheet contains a set of template rules. A template rule has two parts: a pattern which is matched against nodes in the source tree and a template which can be instantiated to form part of the result tree. This allows a stylesheet to be applicable to a wide class of documents that have similar source tree structures.

A template is instantiated for a particular source element to create part of the result tree. A template can contain elements that specify literal result element structure. A template can also contain elements from the XSLT namespace that are instructions for creating result tree fragments. When a template is instantiated, each instruction is executed and replaced by the result tree fragment that it creates. Instructions can select and process descendant source elements. Processing a descendant element creates a result tree fragment by finding the applicable template rule and instantiating its template. Note that elements are only processed when they have been selected by the execution of an instruction. The result tree is constructed by finding the template rule for the root node and instantiating its template.

A software module called an XSL processor is used to read XML documents and transform them into other XML documents with different styles.

The C implementation of the XSL processor followed the XSL Transformations standard (version 1.0, November 16, 1999) and included the required behavior of an XSL processor as specified in the XSLT specification.

xslprocess()

Processes XSL Stylesheet with XML document source and Returns success or an error code.

Syntax

```
uword xslprocess(xmlctx *docctx, xmlctx *xslctx,
                 xmlctx *resctx, xmlnode **result);
```

Parameter	IN/OUT	Description
xmlctx	IN/OUT	The XML document context
xslctx	IN	The XSL stylesheet context
resctx	IN	The result document fragment context
result	IN/OUT	The result document fragment node

W3C SAX APIs

SAX is a standard interface for event-based XML parsing, developed collaboratively by the members of the XML-DEV mailing list.

There are two major types of XML (or SGML) APIs:

- tree-based APIs, and
- event-based APIs.

A tree-based API compiles an XML document into an internal tree structure, then allows an application to navigate that tree using the Document Object Model (DOM), a standard tree-based API for XML and HTML documents.

An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface.

Tree-based APIs are useful for a wide range of applications, but they often put a great strain on system resources, especially if the document is large (under very controlled circumstances, it is possible to construct the tree in a lazy fashion to avoid some of this problem). Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

In both of these cases, an event-based API provides a simpler, lower-level access to an XML document: you can parse documents much larger than your available system memory, and you can construct your own data structures using your callback event handlers.

To use SAX, an xmlsaxcb structure is initialized with function pointers and passed to the `xmlinit` call. A pointer to a user-defined context structure may also be included; that context pointer will be passed to each SAX function.

The SAX callback structure:

```
typedef struct
{
   sword (*startDocument)(void *ctx);
   sword (*endDocument)(void *ctx);
   sword (*startElement)(void *ctx, const oratext *name,
                         const struct xmlarray *attrs);
   sword (*endElement)(void *ctx, const oratext *name);
```

```
   sword (*characters)(void *ctx, const oratext *ch, size_t len);
   sword (*ignorableWhitespace)(void *ctx, const oratext *ch, size_t len);
   sword (*processingInstruction)(void *ctx, const oratext *target,
                                  const oratext *data);
   sword (*notationDecl)(void *ctx, const oratext *name,
                         const oratext *publicId, const oratext *systemId);
   sword (*unparsedEntityDecl)(void *ctx, const oratext *name,
                               const oratext *publicId,
                               const oratext *systemId,
                               const oratext *notationName);
   sword (*nsStartElement)(void *ctx, const oratext *qname,
                           const oratext *local, const oratext *nsp,
                           const struct xmlnodes *attrs);
} xmlsaxcb;
```

Table 14–2 summarizes the callback functions. Note that all functions with the exception of `nsStartElement()` conform to the SAX standard.

Table 14–2 Summary of SAX Callback functions.

Method	Description
characters() on page 14-15	Receive notification of character data inside an element.
endDocument() on page 14-15	Receive notification of the end of the document.
endElement() on page 14-15	Receive notification of the end of an element.
ignorableWhitespace() on page 14-16	Receive notification of ignorable whitespace in element content.
notationDecl () on page 14-16	Receive notification of a notation declaration.
processingInstruction() on page 14-17	Receive notification of a processing instruction
startDocument() on page 14-17	Receive notification of the beginning of the document.
startElement() on page 14-17	Receive notification of the start of an element.
unparsedEntityDecl() on page 14-18	Receive notification of an unparsed entity declaration.
nsStartElement() on page 14-18	Receive notification of the start of a namespace for an element.

characters()

This callback function receives notification of character data inside an element.

Syntax

```
sword (*characters)(
   void *ctx,
   const oratext *ch,
   size_t len);
```

Parameter	IN/OUT	Description
ctx	IN	client context pointer
ch	IN	the characters
len	IN	number of characters to use from the character pointer

endDocument()

This callback function receives notification of the end of the document.

Syntax

```
sword (*endDocument)(
   void *ctx);
```

Parameter	IN/OUT	Description
ctx	IN	client context

endElement()

This callback function receives notification of the end of an element.

Syntax

```
sword (*endElement)(
   void *ctx,
   const oratext *name);
```

Parameter	IN/OUT	Description
ctx	IN	client context
name	IN	element type name

ignorableWhitespace()

This callback function receives notification of ignorable whitespace in element content.

Syntax

```
sword (*ignorableWhitespace)(
   void *ctx,
   const oratext *ch,
   size_t len);
```

Parameter	IN/OUT	Description
ctx	IN	client context
ch	IN	whitespace characters
len	IN	number of characters to use from the character pointer

notationDecl ()

Purpose

This callback function receives notification of a notation declaration.

Syntax

```
sword (*notationDecl)(
   void *ctx,
   const oratext *name,
   const oratext *publicId,
   const oratext *systemId);
```

Parameter	IN/OUT	Description
ctx	IN	client context

Parameter	IN/OUT	Description
name	IN	notation name
publicId	IN	notation public identifier, or null if not available
systemId	IN	notation system identifier

processingInstruction()

This callback function receives notification of a processing instruction.

Syntax

```
sword (*processingInstruction)(
   void *ctx,
   const oratext *target,
   const oratext *data);
```

Parameter	IN/OUT	Description
ctx	IN	client context
target	IN	processing instruction target
data	IN	processing instruction data, or null if none is supplied

startDocument()

This callback function receives notification of the beginning of the document.

Syntax

```
sword (*startDocument)(
   void *ctx);
```

Parameter	IN/OUT	Description
ctx	IN	client context

startElement()

This callback function receives notification of the beginning of an element.

Syntax

```
word (*startElement)(
   void *ctx,
   const oratext *name,
   const struct xmlattrs *attrs);
```

Parameter	IN/OUT	Description
ctx	IN	client context
name	IN	element type name
attrs	IN	specified or defaulted attributes

unparsedEntityDecl()

This callback function receives notification of an unparsed entity declaration.

Syntax

```
sword (*unparsedEntityDecl)(
   void *ctx,
   const oratext *name,
   const oratext *publicId,
   const oratext *systemId,
   const oratext *notationName);
```

Parameter	IN/OUT	Description
ctx	IN	client context
name	IN	entity name
publicId	IN	entity public identifier, or null if not available
systemId	IN	entity system identifier
notationName	IN	name of the associated notation

nsStartElement()

This callback function receives notification of the start of a namespace for an element.

Syntax

```
sword (*nsStartElement)(
   void *ctx,
   const oratext *qname,
   const oratext *local,
   const oratext *namespace,
   const struct xmlattrs *attrs);
```

Parameter	IN/OUT	Description
ctx	IN	client context
qname	IN	element fully qualified name
local	IN	element local name
namespace	IN	element namespace (URI)
attrs	IN	specified or defaulted attributes

W3C DOM APIs

The ***Document Object Model*** (**DOM**) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term *document* is used in the broad sense -- increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

With the DOM, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the DOM, with a few exceptions -- in particular, the DOM interfaces for the XML internal and external subsets have not yet been specified.

One important objective of the W3C specification for the DOM is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM is designed to be used with any programming language. Since the DOM standard is object-oriented, for the C adaptation, some changes had to be made:

- Reused function names had to be expanded, e.g. `getValue` in the attribute class is given the unique name `getAttrValue`, matching the pattern established by `getNodeValue`.
- Also, some functions were added to extend the DOM. For example, there is no function defined which Returns the number of children of a node, so `numChildNodes` was invented, etc.

The implementation of this C DOM interface follows REC-DOM-Level-1-19981001. Table 14–3 summarizes the C DOM functions.

Table 14–3 Summary of W3C DOM Methods

Method	Description
appendChild() on page 14-24	Append child node to current node
appendData() on page 14-24	Append character data to end of node's current data
cloneNode() on page 14-24	Create a new node identical to the current one
createAttribute() on page 14-25	Create an new attribute for an element node

Table 14–3 (Cont.) Summary of W3C DOM Methods

Method	Description
createCDATASection() on page 14-25	Create a CDATA node
createComment() on page 14-26	Create a comment node
createDocumentFragment() on page 14-26	Create a document fragment node
createElement() on page 14-26	Create an element node
createEntityReference() on page 14-27	Create an entity reference node
createProcessingInstruction() on page 14-27	Create a processing instruction (PI) node
createTextNode() on page 14-27	Create a text ode
deleteData() on page 14-28	Remove substring from a node's character data
getAttribute() on page 14-28	Return an attribute's name
getAttributeIndex() on page 14-29	Return value of attribute's *specified* flag.
getAttributeNode() on page 14-29	Return an attribute's value (definition).
getAttributes() on page 14-29	Return the value of an attribute
getAttrName() on page 14-30	Return an element's attribute given its index
getAttrSpecified() on page 14-30	Get an element's attribute node given its name.
getAttrValue() on page 14-30	Return array of element's attributes
getCharData() on page 14-31	Return character data for a TEXT node.
getCharLength() on page 14-31	Return length of TEXT node's character data.
getChildNode() on page 14-31	Return indexed node from array of nodes.
getChildNodes() on page 14-32	Return array of node's children
getContentModel() on page 14-32	Returns the content model for an element from the DTD.
getDocType() on page 14-32	Return top-level DOCUMENT node.
getDocTypeEntities() on page 14-33	Return highest-level (root) ELEMENT node.
getDocTypeName() on page 14-33	Return current DTD.

Table 14–3 (Cont.) Summary of W3C DOM Methods

Method	Description
getDocTypeNotations() on page 14-33	Return array of DTD's general entities
getElementsByTagName() on page 14-34	Return name of DTD
getDocument() on page 14-34	Return array of DTD's notations
getDocumentElement() on page 14-34	Return list of elements with matching name
getEntityNotation() on page 14-35	Return an entity's NDATA.
getEntityPubID() on page 14-35	Return an entity's public ID.
getEntitySysID() on page 14-35	Return an entity's system ID.
getFirstChild() on page 14-36	Return the first child of a node
getImplementation() on page 14-36	Return DOM-implementation structure (if defined)
getLastChild() on page 14-36	Return the last child of a node
getNamedItem() on page 14-37	Return a node's next sibling
getNextSibling() on page 14-37	Returns the named node from a list of nodes
getNodeMapLength() on page 14-37	Returns number of entries in a NodeMap.
getNodeName() on page 14-38	Returns a node's name
getNodeType() on page 14-38	Returns a node's type code (enumeration)
getNodeValue() on page 14-38	Returns a node's "value", its character data
getNotationPubID() on page 14-39	Returns a notation's public ID.
getNotationSysID() on page 14-39	Returns a notation's system ID.
getOwnerDocument() on page 14-39	Returns the DOCUMENT node containing the given node
getParentNode() on page 14-40	Returns a processing instruction's data.
getPIData() on page 14-40	Returns a processing instruction's target.
getPITarget() on page 14-40	Returns a node's parent node
getPreviousSibling() on page 14-41	Returns a node's "previous" sibling
getTagName() on page 14-41	Returns a node's "gagman", same as name for now
hasAttributes() on page 14-41	Determine if element node has attributes

Table 14–3 (Cont.) Summary of W3C DOM Methods

Method	Description
hasChildNodes() on page 14-42	Determine if node has children
hasFeature() on page 14-42	Determine if DOM implementation supports a specific feature
insertBefore() on page 14-43	Inserts a new child node before the given reference node
insertData() on page 14-43	Inserts new character data into a node's existing data
isStandalone() on page 14-44	Determine if document is standalone
nodeValid() on page 14-44	Validate a node against the current DTD
normalize() on page 14-44	Normalize a node by merging adjacent TEXT nodes
numAttributes() on page 14-45	Returns number of element node's attributes
numChildNodes() on page 14-45	Returns number of node's children
removeAttribute() on page 14-45	Removes an element's attribute given its names
removeAttributeNode() on page 14-46	Removes an element's attribute given its pointer
removeChild() on page 14-46	Removes a node from its parents list of children
removeNamedItem() on page 14-46	Removes a node from a list of nodes given its name
replaceChild() on page 14-47	Replace one node with another
replaceData() on page 14-47	Replace a substring of a node's character data with another string
setAttribute() on page 14-48	Sets (adds or replaces) a new attribute for an element node given the attribute's name and value
setAttributeNode() on page 14-48	Sets (adds or replaces) a new attribute for an element node given a pointer to the new attribute
setNamedItem() on page 14-49	Sets (adds or replaces) a new node in a parent's list of children
setNodeValue() on page 14-49	Sets a node's "value" (character data)
setPIData() on page 14-49	Sets a processing instruction's data
setPIData() on page 14-49	Split a node's character data into two parts
substringData() on page 14-50	Return a substring of a node's character data

appendChild()

Adds new node to the end of the list of children for the given parent and Returns the node added.

Syntax

```
xmlnode *appendChild(xmlctx *ctx, xmlnode *parent, xmlnode *newnode)
```

Parameter	IN/OUT	Description
ctx	(IN)	XML context
parent	(IN)	parent node
newnode	(IN)	new node to append
newnode	(IN)	new node to append

appendData()

Append the given string to the character data of a TEXT or CDATA node.

Syntax

```
void appendData(xmlctx *ctx, xmlnode *node, const oratext *arg)
```

Parameter	IN/OUT	Description
ctx	(IN)	XML context
node	(IN)	pointer to node
arg	(IN)	new data to append

cloneNode()

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` Returns `NULL`).

Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child Text node. Cloning any other type of node simply Returns a copy of this node.

A *deep* clone differs in that the node's children are also recursively cloned instead of just pointed-to.

Syntax

```
xmlnode *cloneNode(xmlctx *ctx, const xmlnode *old, boolean deep)
```

Parameter	IN/OUT	Description
ctx	(IN)	XML context
old	IN	old node to clone
deep	(IN)	recursion flag

createAttribute()

Create a new ATTRIBUTE node with the given name and value. The new node is unattached and must be added to an element node with `setAttributeNode`.

Syntax

```
xmlnode *createAttribute(xmlctx *ctx, const oratext *name, const oratext *value)
```

Parameter	IN/OUT	Description
ctx	(IN)	XML context
name	(IN)	name of new attribute
value	IN	value of new attribute

createCDATASection()

Create a new CDATA node.

Syntax

```
xmlnode *createCDATASection(xmlctx *ctx, const oratext *data)
```

Parameter	IN/OUT	Description
ctx	(IN)	XML context

Parameter	IN/OUT	Description
data	IN	CData body

createComment()

Create a new COMMENT node.

Syntax

```
xmlnode *createComment(xmlctx *ctx, const oratext *data)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
data	IN	text of comment

createDocumentFragment()

Create a new DOCUMENT_FRAGMENT node. A document fragment is a lightweight document object that contains one or more children, but does not have the overhead of a full document. It can be used in some operations (inserting for example) in place of a simple node, in which case all the fragment's children are operated on instead of the fragment node itself.

Syntax

```
xmlnode *createDocumentFragment(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML context

createElement()

Create a new ELEMENT node.

Syntax

```
xmlnode *createElement(xmlctx *ctx, const oratext *elname)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
elname	IN	name of new element

createEntityReference()

Create a new ENTITY_REFERENCE node.

Syntax

```
xmlnode *createEntityReference(xmlctx *ctx, const oratext *name)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
name	IN	name of entity reference

createProcessingInstruction()

Create a new PROCESSING_INSTRUCTION node with the given target and contents.

Syntax

```
xmlnode *createProcessingInstruction(xmlctx *ctx, const oratext *target,
                                     const oratext *data)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
target	IN	PI target
data	IN	PI definition

createTextNode()

Create a new TEXT node with the given contents.

Syntax

```
xmlnode *createTextNode(xmlctx *ctx, const oratext *data)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
data	IN	data for node

deleteData()

Delete a substring from the node's character data.

Syntax

```
void deleteData(xmlctx *ctx, xmlnode *node, ub4 offset, ub4 count)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
node	IN	pointer to Node
offset	IN	offset of start of substring (0 for first char)
count	IN	length of substring

getAttribute()

Returns one attribute from an array of attributes, given an index (starting at 0). Fetch the attribute name and/or value (with getAttrName and getAttrValue). On error, Returns NULL.

Syntax

```
const oratext *getAttribute(const xmlnode *node, const oratext *name)
```

Parameter	IN/OUT	Description
node	IN	Node whose attributes are scanned
name	IN	name of the attribute

getAttributeIndex()

Returns one attribute from an array of attributes, given an index (starting at 0). Fetch the attribute name and/or value (with `getAttrName` and `getAttrValue`). On error, Returns `NULL`.

Syntax

```
xmlnode *getAttributeIndex(const xmlnodes *attrs, size_t index)
```

Parameter	IN/OUT	Description
`attrs`	`IN`	pointer to attribute Node structure
`index`	`IN`	Zero-based attribute number to return

getAttributeNode()

Returns a pointer to the element node's attribute of the given name. If no such thing exists, Returns `NULL`.

Syntax

```
xmlnode *getAttributeNode(const xmlnode *elem, const oratext *name)
```

Parameter	IN/OUT	Description
`elem`	`IN`	pointer to Element node
`name`	`IN`	name of attribute

getAttributes()

Returns an array of all attributes of the given node. This pointer may then be passed to `getAttribute` to fetch individual attribute pointers, or to `numAttributes` to return the total number of attributes. If no attributes are defined, Returns `NULL`.

Syntax

```
xmlnodes *getAttributes(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	the node whose attributes are returned

getAttrName()

Given a pointer to an attribute, Returns the name of the attribute. Under the DOM spec, this is a method named getName.

Syntax

```
const oratext *getAttrName(const xmlnode *attr)
```

Parameter	IN/OUT	Description
attr	IN	pointer to attribute

getAttrSpecified()

Return the 'specified' flag for the attribute: if this attribute was explicitly given a value in the original document or through the DOM, this is TRUE; otherwise, it is FALSE. If the node is not an attribute, Returns FALSE. Under the DOM spec, this is a method named getSpecified.

Syntax

```
boolean getAttrSpecified(const xmlnode *attr)
```

Parameter	IN/OUT	Description
attr	IN	pointer to attribute

getAttrValue()

Given a pointer to an attribute, Returns the "value" (definition) of the attribute. Under the DOM spec, this is a method named getValue.

Syntax

```
const oratext *getAttrValue(const xmlnode *attr)
```

Parameter	IN/OUT	Description
attr	IN	pointer to attribute

getCharData()

Returns the character data of a TEXT or CDATA node. Under the DOM spec, this is a method named `getData`.

Syntax

```
const oratext *getCharData(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getCharLength()

Returns the length of the character data of a TEXT or CDATA node. Under the DOM spec, this is a method named `getLength`.

Syntax

```
ub4 getCharLength(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getChildNode()

Returns the *n*th node in an array of nodes, or `NULL` if the numbered node does not exist. Invented function, not in DOM, but named to match the DOM pattern.

Syntax

```
xmlnode* getChildNode(const xmlnodes *nodes, size_t index)
```

Parameter	IN/OUT	Description
nodes	IN	array of nodes
index	IN	zero-based child number

getChildNodes()

Returns the array of children of the given node. This pointer may then be passed to `getChildNode` to fetch individual children.

Syntax

```
xmlnodes* getChildNodes(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	node whose children are returned

getContentModel()

Returns the content model for the named element from the current DTD. The content model is composed of xmlnodes, so may be traversed with the same functions as the parsed document. See also the `getModifier` function which Returns the '?', '*', and '+' modifiers to content model nodes.

Syntax

```
xmlnode *LpxGetContentModel(xmldtd *dtd, oratext *name)
```

Parameter	IN/OUT	Description
dtd	IN	pointer to the DTD
name	IN	name of element

getDocType()

Returns a pointer to the (opaque) DTD for the current document.

Syntax

```
xmldtd* getDocType(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context

getDocTypeEntities()

Returns an array of (general) entities defined for the given DTD.

Syntax

```
xmlnodes *getDocTypeEntities(xmldtd* dtd)
```

Parameter	IN/OUT	Description
dtd	IN	pointer to the DTD

getDocTypeName()

Returns the given DTD's name.

Syntax

```
oratext *getDocTypeName(xmldtd* dtd)
```

Parameter	IN/OUT	Description
dtd	IN	pointer to the DTD

getDocTypeNotations()

Returns an array of notations defined for the given DTD.

Syntax

```
xmlnodes *getDocTypeNotations(xmldtd* dtd)
```

Parameter	IN/OUT	Description
dtd	IN	pointer to the DTD

getElementsByTagName()

Returns a list of all elements (within the tree rooted at the given node) with a given tag name in the order in which they would be encountered in a pre-order traversal of the tree. If root is NULL, the entire document is searched. The special value "*" matches all tags.

Syntax

```
xmlnodes *getElementsByTagName(xmlctx *ctx, xmlnode *root, const oratext *name)
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context
root	IN	root node
name	IN	element tag name

getDocument()

Returns the root node of the parsed document. The root node is always of type DOCUMENT_NODE. Compare to the getDocumentElement function, which Returns the root *element* node, which is a child of the DOCUMENT node.

Syntax

```
xmlnode* getDocument(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context

getDocumentElement()

Returns the root element (node) of the parsed document. The entire document is rooted at this node. Compare to getDocument which Returns the uppermost DOCUMENT node (the parent of the root element node).

Syntax

```
xmlnode* getDocumentElement(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context

getEntityNotation()

Returns an entity node's NDATA (notation). Under the DOM spec, this is a method named `getNotationName`.

Syntax

```
const oratext *getEntityNotation(const xmlnode *ent)
```

Parameter	IN/OUT	Description
ent	IN	pointer to entity

getEntityPubID()

Returns an entity node's public ID. Under the DOM spec, this is a method named `getPublicId`.

Syntax

```
const oratext *getEntityPubID(const xmlnode *ent)
```

Parameter	IN/OUT	Description
ent	IN	pointer to entity

getEntitySysID()

Returns an entity node's system ID. Under the DOM spec, this is a method named `getSystemId`.

Syntax

```
const oratext *getEntitySysID(const xmlnode *ent)
```

Parameter	IN/OUT	Description
ent	IN	pointer to entity

getFirstChild()

Returns the first child of the given node, or NULL if the node has no children.

Syntax

```
xmlnode* getFirstChild(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getImplementation()

This function returns a pointer to the DOMImplementation structure for this implementation, or NULL if no such information is available.

Syntax

```
xmldomimp* getImplementation(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML context

getLastChild()

Returns the last child of the given node, or NULL if the node has no children.

Syntax

```
xmlnode* getLastChild(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getNamedItem()

Returns the named node from an array nodes; sets the user's index (if provided) to the child# of the node (first node is zero).

Syntax

```
xmlnode *getNamedItem(const xmlnodes *nodes, const oratext *name, size_t *index)
```

Parameter	IN/OUT	Description
nodes	IN	array of nodes
name	IN	name of the node to fetch
index	OUT	index of found node

getNextSibling()

This function returns a pointer to the next sibling of the given node, that is, the next child of the parent. For the last child, NULL is returned.

Syntax

```
xmlnode* getNextSibling(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getNodeMapLength()

Given an array of nodes (as returned by getChildNodes), Returns the number of nodes in the map. Under the DOM spec, this is a member function named getLength.

Syntax

```
size_t getNodeMapLength(const xmlnodes *nodes)
```

Parameter	IN/OUT	Description
nodes	IN	array of nodes

getNodeName()

Returns the name of the given node, or NULL if the node has no name. Note that "getName" and "name" are currently synonymous.

Syntax

```
const oratext* getNodeName(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getNodeType()

Returns the type code for a node.

Syntax

```
xmlntype getNodeType(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getNodeValue()

Returns the "value" (associated character data) for a node, or NULL if the node has no data.

Syntax

```
const oratext* getNodeValue(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getNotationPubID()

Return a notation node's public ID. Under the DOM spec, this is a method named `getPublicId`.

Syntax

```
const oratext *getNotationPubID(const xmlnode *note)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getNotationSysID()

Return a notation node's system ID. Under the DOM spec, this is a method named `getSystemId`.

Syntax

```
const oratext *getNotationSysID(const xmlnode *note)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getOwnerDocument()

Returns the document node which contains the given node. An XML document is always rooted in a node of type DOCUMENT_NODE. Calling `getOwnerDocument` on any node in the document Returns that document node.

Syntax

```
xmlnode* getOwnerDocument(xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getParentNode()

Returns the parent node of the given node. For the top-most node, NULL is returned.

Syntax

```
xmlnode* getParentNode(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getPIData()

Returns a Processing Instruction's (PI) data string. Under the DOM spec, this is a method named getData.

Syntax

```
const oratext *getPIData(const xmlnode *pi)
```

Parameter	IN/OUT	Description
pi	IN	pointer to PI node

getPITarget()

Returns a Processing Instruction's (PI) target string. Under the DOM spec, this is a method named getTarget.

Syntax

```
const oratext *getPITarget(const xmlnode *pi)
```

Parameter	IN/OUT	Description
pi	IN	pointer to PI node

getPreviousSibling()

Returns the previous sibling of the given node. That is, the node at the same level which came before this one. For the first child of a node, NULL is returned.

Syntax

```
xmlnode* getPreviousSibling(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

getTagName()

Returns the "tagname" of a node, which is the same as its name for now, see getNodeName. The DOM says "...even though there is a generic nodeName attribute on the Node interface, there is still a tagName attribute on the Element interface; these two attributes must contain the same value, but the Working Group considers it worthwhile to support both, given the different constituencies the DOM API must satisfy."

Syntax

```
const oratext *getTagName(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

hasAttributes()

Determines if the given node has any defined attributes, returning TRUE if so, FALSE if not. This is a DOM extension named after the pattern started by hasChildNodes.

Syntax

```
boolean hasAttributes(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

hasChildNodes()

Determines if the given node has children, returning TRUE if so, FALSE if not. The same result can be achieved by testing if getChildNodes Returns a pointer (has children) or NULL (no children).

Syntax

```
boolean hasChildNodes(const xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	pointer to node

hasFeature()

Tests if the DOM implementation implements a specific feature and version. *feature* is the package name of the feature to test. In DOM Level 1, the legal values are "HTML" and "XML" (case-insensitive). *version* is the version number of the package name to test. In DOM Level 1, this is the string "1.0". If the version is not specified, supporting any version of the feature will cause the method to return TRUE.

Syntax

```
boolean hasFeature(xmlctx *ctx, const oratext *feature, const oratext *version)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
featue	IN	the name of the package
version	IN	version number of package

insertBefore()

Inserts a new node into the given parent node's list of children before the existing reference node. If the reference node is NULL, appends the new node at the end of the list. If the new node is a DocumentFragment, its children are inserted, in the same order, instead of the fragment itself. If the new node is already in the tree, it is first removed.

Syntax

```
xmlnode *insertBefore(xmlctx *ctx, xmlnode *parent,
                      xmlnode *newChild, xmlnode *refChild)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
parent	IN	parent node
newChild	IN	new child node
refChild	IN	reference node; the inserted node will come before it

insertData()

Inserts a string into the node character data at the specified offset.

Syntax

```
void insertData(xmlctx *ctx, xmlnode *node, ub4 offset, const oratext *arg)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
node	IN	pointer to node
offset	IN	insertion point (first position)
refChild	IN	new string to insert

isStandalone()

Returns the value of the `standalone` flag as specified in the document's <?xml?> processing instruction. This is an invented function, not in DOM spec, but named to match the DOM pattern.

Syntax

```
boolean isStandalone(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML context

nodeValid()

Validate a node against the DTD. Returns 0 on success, else a non-zero error code (which can be looked up in the message file). This function is provided for applications which construct their own documents via the API and/or Class Generator. Normally the parser will validate the document and the user need not call `nodeValid` explicitly.

Syntax

```
uword nodeValid(xmlctx *ctx, const xmlnode *node)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
node	IN	pointer to node

normalize()

Normalizes an element, i.e. merges adjacent TEXT nodes. Adjacent TEXT nodes don't happen during a normal parse, only when extra nodes are inserted via the DOM.

Syntax

```
void normalize(xmlctx *ctx, xmlnode *elem)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
elem	IN	pointer to element node

numAttributes()

Returns the number of defined attributes in an attribute array (as returned by `getAttributes`). This is an invented function, not in the DOM spec, but named after the DOM pattern.

Syntax

```
size_t numAttributes(const xmlnodes *attrs)
```

Parameter	IN/OUT	Description
attrs	IN	array of attributes

numChildNodes()

Returns the number of children in an array of nodes (as returned by `getChildNodes`). This is an invented function, not in the DOM spec, but named after the DOM pattern.

Syntax

```
size_t numChildNodes(const xmlnodes *nodes)
```

Parameter	IN/OUT	Description
nodes	IN	pointer to opaque node structure

removeAttribute()

Removes the named attribute from an element node. If the removed attribute has a default value it is immediately replaced.

Syntax

```
void removeAttribute(xmlnode *elem, const oratext *name)
```

Parameter	IN/OUT	Description
elem	IN	pointer to element node
name	IN	name of attribute to remove

removeAttributeNode()

Removes an attribute from an element, given a pointer to the attribute. If successful, Returns the attribute node back. On error, Returns NULL.

Syntax

```
xmlnode *removeAttributeNode(xmlnode *elem, xmlnode *attr)
```

Parameter	IN/OUT	Description
elem	IN	pointer to element node
attr	IN	attribute node to remove

removeChild()

Removes the given node from its parent and Returns it.

Syntax

```
xmlnode *removeChild(xmlnode *node)
```

Parameter	IN/OUT	Description
node	IN	old node to remove

removeNamedItem()

Removes the named node from an array of nodes.

Syntax

```
xmlnode *removeNamedItem(xmlnodes *nodes, const oratext *name)
```

Parameter	IN/OUT	Description
nodes	IN	list of nodes
name	IN	name of node to remove

replaceChild()

Replaces an existing child node with a new node and Returns the old node. If the new node is already in the tree, it is first removed.

Syntax

```
xmlnode *replaceChild(xmlctx *ctx, xmlnode *newChild, xmlnode *oldChild)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
newChild	IN	new replacement node
oldChild	IN	old node being replaced

replaceData()

Replaces the substring at the given character offset and length with a replacement string.

Syntax

```
void replaceData(xmlctx *ctx, xmlnode *node, ub4 offset,
                 ub4 count, oratext *arg)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
node	IN	pointer to node
offset	IN	start of substring to replace; 0 is the first character
count	IN	length of old substring
arg	IN	replacement text

setAttribute()

Create a new attribute for an element. If the named attribute already exists, its value is simply replaced.

Syntax

```
xmlnode *setAttribute(xmlctx *ctx, xmlnode *elem,
                      const oratext *name, const oratext *value)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
elem	IN	pointer to element node
name	IN	name of new attribute
value	IN	value of new attribute

setAttributeNode()

Adds a new attribute to the given element. If the named attribute already exists, it is replaced and the user's old pointer (if provided) is set to the old `attr`. If the attribute is new, it is added and the old pointer is set to `NULL`. Returns a truth value indicating success.

Syntax

```
boolean setAttributeNode(xmlctx *ctx, xmlnode *elem,
                         xmlnode *newNode, xmlnode **oldNode)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
elem	IN	pointer to element node
newNode	IN	pointer to new attribute
oldNode	OUT	return pointer for old attribute

setNamedItem()

Sets a new child node in a parent node's map; if an old node exists with same name, replaces the old node (and sets user's pointer, if provided, to it); if no such named node exists, appends node to map and sets pointer to NULL.

Syntax

```
boolean setNamedItem(xmlctx *ctx, xmlnode *parent, xmlnode *node, xmlnode **old)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
parent	IN	parent to which the new node is added
node	IN	name of new node
old	IN	pointer to replaced node

setNodeValue()

Sets the *value* (character data) associated with a node.

Syntax

```
boolean setNodeValue(xmlnode *node, const oratext *data)
```

Parameter	IN/OUT	Description
node	IN	pointer to node
data	IN	new data for node

setPIData()

Sets a Processing Instruction's (PI) data (equivalent to setNodeValue). It is not permitted to set the data to NULL. Under the DOM spec, this is a method named setData.

Syntax

```
void setPIData(xmlnode *pi, const oratext *data)
```

Parameter	IN/OUT	Description
pi	IN	pointer to PI node
data	IN	new data for PI

splitText()

Breaks a TEXT node into two TEXT nodes at the specified offset, keeping both in the tree as siblings. The original node then only contains all the content up to the offset point. And a new node, which is inserted as the next sibling of the original, contains all the old content starting at the offset point.

Syntax

```
xmlnode *splitText(xmlctx *ctx, xmlnode *old, uword offset)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
old	IN	original node to split
offset	IN	offset of split point

substringData()

Returns a substring of a node's character data.

Syntax

```
const oratext *substringData(xmlctx *ctx, const xmlnode *node,
                             ub4 offset, ub4 count)
```

Parameter	IN/OUT	Description
ctx	IN	XML context
node	IN	pointer to node
offset	IN	start of substring to replace; 0 is the first character
count	IN	length of substring
arg	IN	replacement text

Namespace APIs

Namespace APIs provide an interface that is an extension to the DOM and give information relating to the document namespaces.

XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references. A single XML document may contain elements and attributes (here referred to as a "markup vocabulary") that are defined for and used by multiple software modules. One motivation for this is modularity; if such a markup vocabulary exists which is well-understood and for which there is useful software available, it is better to re-use this markup rather than re-invent it.

Such documents, containing multiple markup vocabularies, pose problems of recognition and collision. Software modules need to be able to recognize the tags and attributes which they are designed to process, even in the face of "collisions" occurring when markup intended for some other software package uses the same element type or attribute name.

These considerations require that document constructs should have universal names, whose scope extends beyond their containing document. This C implementation of XML namespaces provides a mechanism to accomplish this.

Names from XML namespaces may appear as qualified names, which contain a single colon, separating the name into a namespace prefix and a local part. The prefix, which is mapped to a URI reference, selects a namespace. The combination of the universally managed URI namespace and the document's own namespace produces identifiers that are universally unique. Mechanisms are provided for prefix scoping and defaulting.

URI references can contain characters not allowed in names, so cannot be used directly as namespace prefixes. Therefore, the namespace prefix serves as a proxy for a URI reference. An attribute-based Syntax described in the W3C Namespace specification is used to declare the association of the namespace prefix with a URI reference.

The implementation of this C Namespace interface followed the XML Namespace standard of revision REC-xml-names-19990114.

Table 14–4 lists the data structures and types of C Namespace.

Table 14–4 Data Structures and Types of C Namespace

Type	Definition	Description
oratext	typedef unsigned char oratext;	__
xmlattr	typedef struct xmlattr xmlattr;	The contents of xmlattr are private and must not be accessed by users.
xmlnode	typedef struct xmlnode xmlnode;	The contents of xmlnode are private and must not be accessed by users.

Table 14–5 summarizes the methods of C Namespace.

Table 14–5 Summary of Methods C Namespace

Type	Description
getAttrLocal() on page 14-52	Returns attribute local name.
getAttrNamespace() on page 14-53	Returns attribute namespace (URI).
getAttrPrefix() on page 14-53	Returns attribute prefix.
getAttrQualifiedName() on page 14-53	Returns attribute fully qualified name.
getNodeLocal() on page 14-53	Returns node local name.
getNodeNamespace() on page 14-54	Returns node namespace (URI).
getNodePrefix() on page 14-54	Returns node prefix.
getNodeQualifiedName() on page 14-54	Returns node qualified name.

getAttrLocal()

This function returns the local name of this attribute.

Syntax

```
const oratext *getAttrLocal(const xmlattr *attr);
```

Parameter	IN/OUT	Description
attr	IN	pointer to opaque attribute structure (see getAttribute)

getAttrNamespace()

This function returns namespace for this attribute.

Syntax

```
const oratext *getAttrNamespace(const xmlattr *attr);
```

Parameter	IN/OUT	Description
attr	IN	pointer to opaque attribute structure (see getAttribute)

getAttrPrefix()

This function returns prefix for this attribute.

Syntax

```
const oratext *getAttrPrefix(const xmlattr *attr);
```

Parameter	IN/OUT	Description
attr	IN	pointer to opaque attribute structure (see getAttribute)

getAttrQualifiedName()

This function returns fully qualified name for the attribute.

Syntax

```
const oratext *getAttrQualifiedName(const xmlattr *attr);
```

Parameter	IN/OUT	Description
attr	IN	pointer to opaque attribute structure (see getAttribute)

getNodeLocal()

This function returns the local name of this node.

Syntax

```
const oratext *getNodeLocal(const xmlnode *node);
```

Parameter	IN/OUT	Description
node	IN	node to get local name from

getNodeNamespace()

This function returns namespace for this node.

Syntax

```
const oratext *getNodeNamespace(const xmlnode *node);
```

Parameter	IN/OUT	Description
node	IN	node to get namespace from

getNodePrefix()

This function returns prefix for this node.

Syntax

```
const oratext *getNodePrefix(const xmlnode *node);
```

Parameter	IN/OUT	Description
node	IN	node to get prefix from

getNodeQualifiedName()

This function returns fully qualified name for this node.

Syntax

```
const oratext *getNodeQualifiedName(const xmlnode *node);
```

Parameter	IN/OUT	Description
node	(N	node to get name from

Datatypes

Table 14–6 lists datatypes.

Table 14–6 Datatypes

Type	Definition	Description
`oratext`	`typedef unsigned char oratext;`	The basic character pointer type.
`String`	`typedef unsigned char String;`	The basic character pointer type.
`xmlctx`	`typedef struct xmlctx xmlctx;`	The top-level XML context. he contents of xmlctx are private and must not be accessed by users.

`xmlmemcb`	`struct xmlmemcb`
	`{`
	`  void *(*alloc)(void *ctx, size_t size);`
	`  void  (*free)(void *ctx, void *ptr);`
	`  void *(*realloc)(void *ctx, void *ptr, size_t size);`
	`};`
	`typedef struct xmlmemcb xmlmemcb`
		The memory callback structure passed to `xmlinit`.

Table 14–6 (Cont.) Datatypes

Type	Definition	Description

xmlsaxcb	struct xmlsaxcb
	{
	sword (*startDocument)(void *ctx);
	sword (*endDocument)(void *ctx);
	sword (*startElement)(void *ctx,
	const oratext *name, const struct xmlattrs *attrs);
	sword (*endElement)(void *ctx, const oratext *name);
	sword (*characters)(void *ctx, const oratext *ch,
	size_t len);
	sword (*ignorableWhitespace)(void *ctx,
	const oratext *ch, size_t len);
	sword (*processingInstruction)(void *ctx,
	const oratext *target, const oratext *data);
	sword (*notationDecl)(void *ctx, const oratext *name,
	const oratext *publicId, const oratext *systemId);
	sword (*unparsedEntityDecl)(void *ctx,
	const oratext *name, const oratext *publicId,
	const oratext *systemId,
	const oratext *notationName);
	sword (*nsStartElement)(void *ctx,
	const oratext *qname, const oratext *local,
	const oratext *namespace,
	const struct xmlattrs *attrs);
	};
	typedef struct xmlsaxcb xmlsaxcb;
		The SAX callback structure passed to xmlinit.

ub4	typedef unsigned int ub4;	Unsigned integer with a minimum of four bytes.
uword	typedef unsigned int uword;	Unsigned integer in the native word size
boolean	typedef int boolean;	
oratext	typedef unsigned char oratext;	
xmlcpmod	XMLCPMOD_NONE = 0 /* no modifier */ XMLCPMOD_OPT = 1 /* '?' optional */ XMLCPMOD_0MORE = 2 /* '*' zero or more */ XMLCPMOD_1MORE = 3 /* '+' one or more */	Content model node modifiers, see getModifier.
xmlctx	typedef struct xmlctx xmlctx;	The contents of xmlctx are private and must not be accessed by users.
xmlnode	typedef struct xmlnode xmlnode;	The contents of xmlnode are private and must not be accessed by users.

Table 14–6 (Cont.) Datatypes

Type	Definition	Description
xmlnodes	`typedef struct xmlnodes xmlnodes;`	The contents of xmlnodes are private and must not be accessed by users.

xmlntype	`ELEMENT_NODE = 1 /* element */`
	`ATTRIBUTE_NODE = 2 /* attribute */`
	`TEXT_NODE = 3 /* char data not escaped by CDATA*/`
	`CDATA_SECTION_NODE = 4 /* char data escaped by CDATA*/`
	`ENTITY_REFERENCE_NODE = 5 /* entity reference*/`
	`ENTITY_NODE = 6 /* entity */`
	`PROCESSING_INSTRUCTION_NODE = 7 /* processing instruction */`
	`COMMENT_NODE = 8 /* comment */`
	`DOCUMENT_NODE = 9 /* document */`
	`DOCUMENT_TYPE_NODE = 10 /* DTD */`
	`DOCUMENT_FRAGMENT_NODE = 11 /* document fragment*/`
	`NOTATION_NODE = 12 /* notation */`
		Parse tree node types, see `getNodeType`. Names and values match DOM specification.

Part III

C++ Support for XML

This section contains the following chapters:

- Chapter 15, "XML Schema Processor for C++"
- Chapter 16, "XML Parser for C++"
- Chapter 17, "Oracle XML Class Generator (C++)"

15

XML Schema Processor for C++

The schema API is very simple: initialize, validate,...validate, terminate.

The validation process is go/no-go. Either the document is valid with respect to the schemas or it is invalid. When it is valid, a zero error code is returned. When it is invalid, a non-zero error code is returned indicating the problem. There is no distinction between warnings and errors; all problems are errors and considered fatal: validation stops immediately.

As schemas are encountered, they are loaded and preserved in the schema context. No schema is loaded more than once during a session. There is no clean up call similar to `xmlclean`. Hence, if you need to release all memory and reset state before validating a new document, you must terminate the context and start over.

This chapter contains the following sections:

- XML Schema Methods for C++

See Also:

- *Oracle Application Developer's Guide - XML*

XML Schema Methods for C++

Table 15–1 summarizes the methods of the C++ parser.

Table 15–1 Summary of Methods of XML Schema for C++

Method	Description
XMLSchema::initialize() on page 15-2	Initializes the XML schema processor.
XMLSchema::validate() on page 15-2	Validate an instance document against a schema
XMLSchema::terminate() on page 15-3	Terminates (tears down) the schema processor

XMLSchema::initialize()

Initializes the XML schema processor. Must be called before the processor can be used to validate any documents. The XML parser context is used to allocate memory for the schema context. The schema context is returned, and must be passed to all subsequent schema functions. This context pointer is opaque-- you cannot reference its members. If the return context is NULL, initialization failed and err will be set with the numeric error code indicating the problem.

Syntax

```
uword initialize(xmlctx *ctx)
```

Parameter	IN/OUT	Description
ctx	IN	XML parser context

XMLSchema::validate()

Validates an instance document against a schema or schemas.The schema context returned by XMLSchema::initialize() must be passed in. The document to be validated is specified by the XML parser context used to parse the document. Note the document must already have been parsed. If no schemas are explicitly referenced in the instance document, the default schema (specified by URL) is assumed. If the document *does* specify all necessary schemas, and a default schema

is also supplied, the default will be ignored. If the document does not reference any schemas and no default is supplied, an error will result.

Syntax

```
uword validate(xmlctx *inst, oratext *schema)
```

Parameter	IN/OUT	Description
inst	IN	Instance document context
schema	IN	URL of default schema

Comments

XMLSchema::terminate()

Terminates (shuts down) the schema processor, freeing all memory allocated on the original XML parser context passed to XMLSchema::initialize(). After termination, the schema context is no longer valid. To continue using the schema processor, a new schema must be created with XMLSchema::initialize().

Syntax

```
void terminate()
```

16

XML Parser for C++

This chapter contains the following sections:

See Also:

- *Oracle Application Developer's Guide - XML*

Class Attr

Accessing the name and value of a single document node attribute.

Table 16–1 Summary of Methods of Attr

Method	Description
getName() on page 16-2	Returns name of an attribute.
getValue() on page 16-2	Returns definition of an attribute.
getSpecified() on page 16-2	Returns attribute's "specified" flag value.
setValue() on page 16-3	Sets an attribute's value.

getName()

Returns name of an attribute.

Syntax

```
String getName()
```

getValue()

Returns the definition of an attribute.

Syntax

```
String getValue()
```

getSpecified()

Returns value of attribute's "specified" flag. The DOM specifies"If this attribute was explicitly given a value in the original document, this is true; otherwise, it is false. Note that the implementation is in charge of this attribute, not the user. If the user changes the value of the attribute (even if it ends up having the same value as the default value) then the specified flag is automatically flipped to true. To re-specify the attribute as the default value from the DTD, the user must delete the attribute. The implementation will then make a new attribute available with specified set to false and the default value (if one exists)."

Syntax

```
boolean getSpecified()
```

setValue()

Sets an attribute's "value".

Syntax

```
void setValue(String value)
```

Parameter	Description
value	Attribute's new value

Class CDATASection

This class implements the CDATA node type, a subclass of **Text**.

Class Comment

This class implements the COMMENT node type, a subclass of **CharacterData**.

Class Document

This class contains methods for creating and retrieving nodes.

Table 16–2 Summary of Methods of Document

Method	Description
createAttribute() on page 16-6	Create an ATTRIBUTE node.
createCDATASection() on page 16-7	Create a CDATA node.
createComment() on page 16-7	Create a COMMENT node.
createDocumentFragment() on page 16-7	Create a DOCUMENT_FRAGMENT node.
createElement() on page 16-7	Create an ELEMENT node.
createEntityReference() on page 16-8	Create an ENTITY_REFERENCE node.
createProcessingInstruction() on page 16-8	Create a PROCESSING_INSTRUCTION node.
createTextNode() on page 16-8	Create a TEXT node.
getElementsByTagName() on page 16-9	Select nodes based on tag name.
getImplementation() on page 16-9	Return DTD for document.

createAttribute()

Create a new attribute node. Use setValue to set its value. Returns a pointer to node.

Syntax

```
Attr* createAttribute(String name)
```

Parameter	Description
name	name of attribute

createCDATASection()

Create a new CDATA node with the given contents. Returns a pointer to node.

Syntax

```
Attr* createCDATASection(String name)
```

Parameter	Description
data	contents of node

createComment()

Create a new comment node with the given contents. Returns a pointer to node.

Syntax

```
Comment* createComment(String data)
```

Parameter	Description
data	contents of node

createDocumentFragment()

Create a new document fragment node. Returns a pointer to created node.

Syntax

```
DocumentFragment* createDocumentFragment()
```

createElement()

Create a new element node with the given (tag) name. Returns a pointer to node.

Syntax

```
Element* createElement(String tagName)
```

Parameter	Description
tagName	element's tag name

createEntityReference()

Create a new entity reference node. Returns a pointer to created node.

Syntax

```
EntityReference* createEntityReference(String name)
```

Parameter	Description
name	name of entity to reference

createProcessingInstruction()

Create a new processing instruction node. Returns a pointer to created node.

Syntax

```
ProcessingInstruction* createProcessingInstruction(String target, String data)
```

Parameter	Description
target	target part of PI
data	data for node

createTextNode()

Create a new TEXT node. Returns a pointer to created node.

Syntax

```
Text* createTextNode(String data)
```

Parameter	Description
data	data for node

getElementsByTagName()

Returns a NodeList of all the Elements with a given tag name in the order in which they would be encountered in a preorder traversal of the Document tree. The special value "*" matches all tags. Returns list of matches, NULL if none.

Syntax

```
NodeList* getElementsByTagName(String tagname)
```

Parameter	Description
tagname	tag name to select

getImplementation()

Returns the DOMImplementation structure, currently useless. Perhaps it will be used in later DOM versions. Returns pointer to structure.

Syntax

```
DOMImplementation* getImplementation()
```

Class DocumentType

This class contains methods for accessing information about the Document Type Definition (DTD) of a document.

Table 16–3 Summary of Methods of DocumentType

Method	Description
getName() on page 16-10	Returns name of DTD.
getEntities() on page 16-10	Returns map of DTD's entities.
getNotations() on page 16-10	Returns map of DTD's notations.

getName()

Return name of DTD.

Syntax

```
String getName()
```

getEntities()

Returns map of DTD's (general) entities.

Syntax

```
NamedNodeMap* getEntities()
```

getNotations()

Returns map of DTD's notations.

Syntax

```
NamedNodeMap* getNotations()
```

Class DOMImplementation

This class contains methods relating to the specific DOM implementation supported by the parser.

hasFeature()

Test if the DOM implementation implements a specific feature. Returns TRUE if the feature is supported.

Syntax

```
boolean hasFeature(DOMString feature, DOMString version)
```

Parameter	Description
feature	The package name of the feature to test. In Level 1, the legal values are “HTML” and “XML” (case-insensitive)
version	This is the version number of the package name to test. In Level 1, this is the string “1.0”. If the version is not specified, supporting any version of the feature will cause the method to return true.

Class Element

This class contains methods pertaining to element nodes.

Table 16–4 Summary of Methods of Element

Method	Description
getTagName() on page 16-12	Return the node's tag name
getAttribute() on page 16-12	Select an attribute given its name
setAttribute() on page 16-13	Create a new attribute given its name and value
removeAttribute() on page 16-13	Remove an attribute given its name
getAttributeNode() on page 16-13	Remove an attribute given its name
setAttributeNode() on page 16-14	Add a new attribute node
removeAttributeNode() on page 16-14	Remove an attribute node
getElementsByTagName() on page 16-14	Return a list of element nodes with the given tag name
normalize() on page 16-15	Normalize an element (merge adjacent text nodes)

getTagName()

Return the tag name of the element. The DOM says: "...even though there is a generic nodeName attribute on the Node interface, there is still a tagName attribute on the Element interface; these two attributes must contain the same value, but the Working Group considers it worthwhile to support both, given the different constituencies the DOM API must satisfy.

Syntax

```
String getTagName()
```

getAttribute()

Return "value" (definition) of named attribute

Syntax

```
String getAttribute(String name)
```

Parameter	Description
name	name of attribute

setAttribute()

Create a new attribute.

Syntax

```
Attr* setAttribute(String name, String value)
```

Parameter	Description
name	name of new attribute
value	value of new attribute

removeAttribute()

Removes the named attribute

Syntax

```
void removeAttribute(String name)
```

Parameter	Description
name	name of attribute to remove

getAttributeNode()

Return pointer to named attribute.

Syntax

```
Attr* getAttributeNode(DOMString name)
```

Parameter	Description
naem	name of attribute

setAttributeNode()

Set (add) new attribute. Returns TRUE on success.

Syntax

```
boolean setAttributeNode(Attr* newAttr, Attr** oldAttr)
```

Parameter	Description
newAttr	pointer to new attribute
oldAttr	returned pointer to replaced attribute

removeAttributeNode()

Remove the named attribute

Syntax

```
Attr* removeAttributeNode(Attr* oldAttr)
```

Parameter	Description
oldAttr	attribute to remove

getElementsByTagName()

Create a list of matching elements.

Syntax

```
NodeList* getElementsByTagName(DOMString name)
```

Parameter	Description
name	tagname to match, "*" for all

normalize()

Normalize an element, i.e. merge all adjacent TEXT nodes.

Syntax

```
void normalize(void)
```

Class Entity

This class implements the ENTITY node type, a subclass of **Node**.

Table 16–5 Summary of Methods of Entity

Method	Description
getNotationName() on page 16-16	Return entity's NDATA (notation name)
getPublicId() on page 16-16	Return entity's public ID
getSystemId() on page 16-16	Return entity's system ID

getNotationName()

Return an entity node's notation name NDATA.

Syntax

```
String* getNotationName()
```

getPublicId()

Return an entity node's public ID.

Syntax

```
String getPublicId()
```

getSystemId()

Return an entity node's system ID.

Syntax

```
String getSystemId()
```

Class EntityReference

This class implements the ENTITY_REFERENCE node type, a subclass of Node.

Class NamedNodeMap

This class contains methods for accessing the number of nodes in a node map and fetching individual nodes.

Table 16–6 Summary of Methods of NamedNodeMap

Method	Description
item() on page 16-18	Return *n*th node in map.
getLength() on page 16-18	Return number of nodes in map
getNamedItem() on page 16-18	Select a node by name
setNamedItem() on page 16-19	Set a node into the map
removeNamedItem() on page 16-19	Remove the named node from map

item()

Return nth node in node map.

Syntax

```
Node* item(size_t index)
```

Parameter	Description
index	zero-based node number

getLength()

Return number of nodes in map.

Syntax

```
size_t getLength()
```

getNamedItem()

Selects the node with the given name from the map.

Syntax

```
Node* getNamedItem(String name)
```

Parameter	Description
name	name of node to select

setNamedItem()

Adds a node to the map replacing any node that already exists with the same name.

Syntax

```
boolean setNamedItem(Node *node, Node **old)
```

Parameter	Description
node	Name of node to add
old	Pointer to replaced node, NULL if node is new

removeNamedItem()

Removes the node with the given name from the node map.

Syntax

```
Node* removeNamedItem(String name)
```

Parameter	Description
name	name of node to remove

Class Node

This class contains methods for details about a document node.

Table 16–7 Summary of Methods of Node

Method	Description
appendChild() on page 16-21	Append a new child to the end of the current node's list of children
cloneNode() on page 16-21	Clone an existing node and optionally all its children
getAttributes() on page 16-22	Return structure contains all defined node attributes
getChildNode() on page 16-22	Return specific indexed child of given node
getChildNodes() on page 16-22	Return structure contains all child nodes of given node
getFirstChild() on page 16-22	Return first child of given node
getLastChild() on page 16-22	Return last child of given node
getLocal() on page 16-22	Returns the local name of the node
getNamespace() on page 16-23	Return a node's namespace
getNextSibling() on page 16-23	Return a node's next sibling
getName() on page 16-23	Return name of node
getType() on page 16-23	Return numeric type-code of node
getValue() on page 16-23	Return "value" (data) of node
getOwnerDocument() on page 16-23	Return document node which contains a node
getParentNode() on page 16-24	Return parent node of given node
getPrefix() on page 16-24	Returns the namespace prefix for the node
getPreviousSibling() on page 16-24	Returns the previous sibling of the current node
getQualifiedName() on page 16-24	Return namespace qualified node of given node
hasAttributes() on page 16-24	Determine if node has any defined attributes
hasChildNodes() on page 16-24	Determine if node has children
insertBefore() on page 16-24	Insert new child node into a node's list of children

Table 16–7 (Cont.) Summary of Methods of Node

Method	Description
numChildNodes() on page 16-25	Return count of number of child nodes.
removeChild() on page 16-25	Removes a child from the current node.
replaceChild() on page 16-25	Replace a child node with another
setValue() on page 16-26	Sets a node's value (data)

appendChild()

Append a new child to the current node's list of children.

Syntax

```
Node* appendChild(Node *newChild)
```

Parameter	Description
newChild	new child node

cloneNode()

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (parentNode Returns NULL). Cloning an Element copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child Text node. Cloning any other type of node simply Returns a copy of this node.

Syntax

```
Node* cloneNode(boolean deep)
```

Parameter	Description
deep	recursion flag

getAttributes()

Return structure of all attributes for node

Syntax

```
NamedNodeMap* getAttributes()
```

getChildNode()

Return one of the node's children.

Syntax

```
Node* getChildNode(uword index)
```

Parameter	Description
index	child number, starting at 0

getChildNodes()

Return node's children.

Syntax

```
NodeList* getChildNodes()
```

getFirstChild()

Return the node's first child.

Syntax

```
Node* getFirstChild()
```

getLastChild()

Return the node's last child.

Syntax

```
Node* getLastChild()
```

getLocal()

Return the node's local name.

Syntax

```
String getLocal()
```

getNamespace()

Return the node's namespace.

Syntax

```
String getNamespace()
```

getNextSibling()

Returns the next sibling of the node.

Syntax

```
Node* getNextSibling()
```

getName()

Return name of node, or NULL if the node has no name.

Syntax

```
String getName()
```

getType()

Return numeric type-code for node. These include ELEMENT_NODE, ATTRIBUTE_NODE, TEXT_NODE, CDATA_SECTION_NODE, ENTITY_REFERENCE_NODE, ENTITY_NODE, PROCESSING_INSTRUCTION_NODE, COMMENT_NODE, DOCUMENT_NODE, DOCUMENT_TYPE_NODE, DOCUMENT_FRAGMENT_NODE, NOTATION_NODE.

Syntax

```
short getType()
```

getValue()

Return "value" (data) of node, or NULL if the node has no value.

Syntax

```
String getValue()
```

getOwnerDocument()

Return document node which contains the current node.

Syntax

```
Document* getOwnerDocument()
```

getParentNode()

Return node's parent.

Syntax

```
Node* getParentNode()
```

getPrefix()

Return the namespace prefix of node.

Syntax

```
String getPrefix()
```

getPreviousSibling()

Returns the previous sibling of the node.

Syntax

```
Node* getPreviousSibling()
```

getQualifiedName()

Return the fully qualified (namespace) name of node.

Syntax

```
String getQualifiedName()
```

hasAttributes()

Determine if node has any defined attributes.

Syntax

```
boolean hasAttributes()
```

hasChildNodes()

Determine if node has any children.

Syntax

```
boolean hasChildNodes()
```

insertBefore()

Insert a new child node before the reference node into the list of children nodes.

Syntax

```
Node* insertBefore(Node *newChild, Node *refChild)
```

Parameter	Description
newChild	new node to insert
refChild	reference node; new node comes before

numChildNodes()

Return count of node's children.

Syntax

```
uword numChildNodes()
```

removeChild()

Removes a child node from the current node's list of children.

Syntax

```
Node* removeChild(Node *oldChild)
```

Parameter	Description
oldChild	old node being removed

replaceChild()

Replaces one node in the list of children with another.

Syntax

```
Node* replaceChild(Node *newChild, Node *oldChild)
```

Parameter	Description
newChild	new replacement node
oldChild	old node being replaced

setValue()

Sets a node's “value” (data)

Syntax

```
void setValue(String data)
```

Parameter	Description
`data`	New data for node

Class NodeList

This class contains methods for extracting nodes from a NodeList

Table 16–8 Summary of Methods of NodeList

Method	Description
item() on page 16-27	Return *n*th node in list.
getLength() on page 16-27	Return number of nodes in list

item()

Return *n*th node in node list.

Syntax

```
Node* item(size_t index)
```

Parameter	Description
index	zero-based node number

getLength()

Return number of nodes in list.

Syntax

```
size_t getLength()
```

Class Notation

This class implements the NOTATION node type, a subclass of **Node**.

Table 16–9 Summary of Methods of Notation

Method	Description
getData() on page 16-28	Return notation's data
getTarget() on page 16-28	Return notation's target
setData() on page 16-28	Set notation's data

getData()

Return a notation's data.

Syntax

```
String getData()
```

getTarget()

Return a notation's target.

Syntax

```
String getTarget()
```

setData()

Set a notation's data

Syntax

```
void setData(String data)
```

Parameter	Description
data	new data

Class ProcessingInstruction

Implements the PROCESSING_INSTRUCTION node type, a subclass of **Node**.

Table 16–10 Summary of Methods of ProcessingInstruction

Method	Description
getData() on page 16-29	Return the PI's data.
getTarget() on page 16-29	Return the PI's target.
setData() on page 16-29	Set the PI's data.

getData()

Return data for a processing instruction.

Syntax

```
String getData()
```

getTarget()

Return a processing instruction's target value.

Syntax

```
String getTarget()
```

setData()

Set the data for a processing instruction.

Syntax

```
void setData(String data)
```

Parameter	Description
data	PI's new data

Class Text

Accesses and modifies the data associated with text nodes (subclasses CharacterData).

splitText()

Split a text node in two. The original node retains its data up to the split point, and the remaining data is turned into a new text node which follows. Returns the pointer to new text node.

Syntax

```
Text* splitText(unsigned long offset)
```

Parameter	Description
`offset`	split point

Class XMLParser

This class contains top-level methods for invoking the parser and returning high-level information about a document.

Table 16–11 Summary of Methods of XMLParser

Method	Description
xmlinit() on page 16-31	Initialize XML parser
xmlterm() on page 16-32	Terminate XML parser
xmlparse() on page 16-32	Parse a document from a file
xmlparseBuffer() on page 16-32	Parse a document from a buffer
getContent() on page 16-33	Returns the content model for an element
getModifier() on page 16-33	Returns the modifier for a content-model node
getDocument() on page 16-34	Returns the root node of a parsed document
getDocumentElement() on page 16-34	Returns the root element of a parsed document
getDocType() on page 16-34	Returns the document type string
isStandalone() on page 16-34	Returns the value of the standalone flag
isSingleChar() on page 16-34	Determine document encoding.
getEncoding() on page 16-35	Return name of document's character encoding.

xmlinit()

Initialize XML parser. Returns error code, 0 on success.

Syntax

```
uword xmlinit(oratext *encoding,
              void (*msghdlr)(void *msgctx, oratext *msg, ub4 errcode),
              void *msgctx, lpxsaxcb *saxcb, void *saxcbctx, oratext *lang)
```

Parameter	Description
encoding	Input file's encoding, default UTF8
msghdlr	Error message callback
msgctx	User-defined context pointer passed to msghdlr
saxcb	SAX callback structure (iff using SAX)
saxcbctx	User-defined SAX context structure passed to SAX callback functions
lang	Language for error message (not used)

xmlterm()

Terminate XML parser, tear down, free memory, and so on.

Syntax

```
void xmlterm()
```

xmlparse()

Parses a document. Returns error code, 0 on success.

Syntax

```
uword xmlparse(oratext *doc, oratext *encoding, ub4 flags)
```

Parameter	Description
doc	document path
encoding	document's encoding
flags	Mask of flag bits: ■ XML_FLAG_VALIDATE -- Validate document against DTD ■ XML_FLAG_DISCARD_WHITESPACE -- Discard ignorable whitespace

xmlparseBuffer()

Parses a document. Returns error code, 0 on success.

Syntax

```
uword xmlparseBuffer(oratext *buffer, size_t len, oratext *encoding, ub4 flags)
```

Parameter	Description
buffer	buffer containing document to parse
len	length of document
encoding	document's encoding
flags	Mask of flag bits ■ XML_FLAG_VALIDATE -- Validate document against DTD ■ XML_FLAG_DISCARD_WHITESPACE -- Discard ignorable whitespace

getContent()

Returns the content model for a node. Content model nodes are Nodes and can be traversed and examined with the same functions as the parsed document.

Syntax

```
Node* getContent(Node *node)
```

Parameter	Description
node	content model node whose modifer to return

getModifier()

Returns the modifier for a content model node. The modifier is one of XMLCPMOD_NONE (no modifier), XMLCPMOD_OPT ('?', optional), XMLCPMOD_0MORE ('*', zero or more), or XMLCPMOD_1MORE ('+', one or more).

Syntax

```
xmlcpmod getContent(Node *node)
```

Parameter	Description
node	content model node whose modifer to return

getDocument()

After a document has been successfully parsed, Returns a pointer to the root node of the document. Compare with `getDocumentElement` which Returns the root *element* node. Pointer to root node of document.

Syntax

```
Node* getDocument()
```

getDocumentElement()

Returns a pointer to the root element (node) of a parsed document.

Syntax

```
Element* getDocumentElement()
```

Returns

`Element*` -- Pointer to root element (node) of document

getDocType()

Returns a pointer to a "DocType" structure which describes the DTD.

Syntax

```
DocumentType* getDocType()
```

isStandalone()

Returns TRUE if the document is specified as standalone on the `<?xml?>` line, FALSE otherwise. Value of standalone flag.

Syntax

```
boolean isStandalone()
```

isSingleChar()

Returns a flag which specifies whether the current document is encoded as single-byte characters, ASCII, or multi-byte characters, UTF-8. Compare to `getEncoding`, which Returns the actual name of the document's encoding.

Syntax

```
boolean isSingleChar()
```

getEncoding()

Returns the name of the current document's character encoding scheme, such as ASCII, UTF8, and so on. Compare to `isSingleChar` which just returns a boolean flag saying whether the current encoding is single or multi-byte.

Syntax

```
String getEncoding()
```

C++ SAX API

The SAX API is based on callbacks. Instead of the entire document being parsed and turned into a data structure which may be referenced (by the DOM interface), the SAX interface is serial. As the document is processed, appropriate SAX user callback functions are invoked. Each callback function returns an error code, zero meaning success, any non-zero value meaning failure. If a non-zero code is returned, document processing is stopped. To use SAX, an `xmlsaxcb` structure is initialized with function pointers and passed to the `xmlinit()` call. A pointer to a user-defined context structure may also be included; that context pointer will be passed to each SAX function.

SAX callback structure

```
typedef struct
{
   sword (*)(void *ctx);
   sword (*)(void *ctx);
   sword (*)(void *ctx, const oratext *name, struct xmlarray *attrs);
   sword (*)(void *ctx, const oratext *name);
   sword (*)(void *ctx, const oratext *ch, size_t len);
   sword (*)(void *ctx, const oratext *ch, size_t len);
   sword (*)(void *ctx, const oratext *target, const oratext *data);
   sword (*)(void *ctx, const oratext *name,
                          const oratext *publicId, const oratext *systemId);
   sword (*)(void *ctx, const oratext *name, const oratext *publicId,
                                 const oratext *systemId, const oratext
*notationName);
   sword (*)(void *ctx, const oratext *qname,
                             const oratext *local, const oratext *namespace);
} xmlsaxcb;
```

Table 16–12 Summary of Methods C++ SAX

Method	Description
startDocument() on page 16-37	Starts document processing.
endDocument() on page 16-37	Ends document processing.
startElement() on page 16-38	Starts processing of each new element.

Table 16–12 (Cont.) Summary of Methods C++ SAX

Method	Description
endElement() on page 16-38	Ends processing of each new element.
characters() on page 16-38	Processes pieces of literal text.
IgnorableWhitespace() on page 16-39	Processes piece of non-significant whitespace.
processingInstruction() on page 16-39	Processes Processing Instructions.
notationDecl() on page 16-40	Processes Notations.
unparsedEntityDecl() on page 16-40	Parses Entities.
nsStartElement() on page 16-41	Starts document processing for documents using explicit namespace.

startDocument()

Called once when document processing is first starting. Returns error code, 0 for success, non-0 for error.

Syntax

```
sword startDocument(void *ctx)
```

Parameter	Description
ctx	User-defined context as passed to initialize()

endDocument()

Called once when document processing is finished. Returns error code, 0 for success, non-0 for error.

Syntax

```
sword endDocument(void *ctx)
```

Parameter	Description
ctx	User-defined context as passed to initialize()

startElement()

Called once for each new document element. Returns error code, 0 for success, non-0 for error.

Syntax

```
sword startElement(void *ctx, const oratext *name, struct xmlarray *attrs)
```

Parameter	Description
ctx	User-defined context as passed to initialize()
name	name of node
attrs	array of node's attributes

endElement()

Called once when each document element closes. Returns error code, 0 for success, non-0 for error.

Syntax

```
sword endElement(void *ctx, const oratext *name)
```

Parameter	Description
ctx	User-defined context as passed to initialize()
name	name of node

characters()

Called for each piece of literal text. Error code, 0 for success, non-0 for error.

Syntax

```
sword characters(void *ctx, const oratext *ch, size_t len)
```

Parameter	Description
ctx	User-defined context as passed to initialize()

Parameter	Description
ch	pointer to text
len	number of character in text

IgnorableWhitespace()

Called for each piece of ignorable (non-significant) whitespace. Returns error code, 0 for success, non-0 for error.

Syntax

```
sword ignorableWhitespace(void *ctx, const oratext *ch, size_t len)
```

Parameter	Description
ctx	User-defined context as passed to initialize()
ch	pointer to whitespace text
len	number of characters of whitespace

processingInstruction()

Called once for each PI (Processing Instruction). Returns error code, 0 for success, non-0 for error.

Syntax

```
sword processingInstruction(void *ctx, const oratext *target,
                            const oratext *data)
```

Parameter	Description
ctx	User-defined context as passed to initialize()
target	PI target
data	PI data

notationDecl()

Called once for each NOTATION. Returns error code, 0 for success, non-0 for error.

Syntax

```
sword notationDecl(void *ctx, const oratext *name,
                   const oratext *publicId, const oratext *systemId)
```

Parameter	Description
ctx	User-defined context as passed to initialize()
name	name of notation
publicId	Public ID
systemId	System ID

unparsedEntityDecl()

Called once for each unparsed entity declaration. Returns error code, 0 for success, non-0 for error.

Syntax

```
sword unparsedEntityDecl( void *ctx, const oratext *name,
                          const oratext *publicId,
                          const oratext *systemId,
                          const oratext *notationName)
```

Parameter	Description
ctx	User-defined context as passed to initialize()
name	name of entity
publicId	Public ID
systemId	System ID
notationName	notation name

nsStartElement()

Namespace variant of startElement: called once for each new document element, when the element uses an explicit namespace. Returns error code, 0 for success, non-0 for error.

Syntax

```
sword startElement(void *ctx, const oratext *qname,
                   const oratext *local, const oratext *namespace)
```

Parameter	Description
ctx	User-defined context as passed to initialize()
qname	qualified namespace
local	umm
namespace	yes, well

17

Oracle XML Class Generator (C++)

This chapter details the XML Class Generator for C++ and how it defines each element.

The XML Class Generator takes a Document Type Definition (DTD) and generates classes for each defined element. Those classes are then used in a C++ program to construct XML documents conforming to the DTD. Supported operating systems are Solaris 2.6, Linux 2.2, and NT 4 (Service Pack 3 and above).

This chapter contains the following sections:

- Using Class Generator for C++
- Class XMLClassGenerator
- Class generated

See Also:

- *Oracle Application Developer's Guide - XML*

Using Class Generator for C++

Input is an XML document containing a DTD. The document body itself is ignored; only the DTD is relevant, though the document must conform to the DTD. The underlying XML parser only accepts file names for the document and associated external entities. The supported input file encodings include UTF-8, UTF-16, US-ASCII, ISO-10646-UCS-2, ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, EUC-JP, SHIFT_JIS, BIG5, GB2312, KOI8-R, EBCDIC-CP-US, EBCDIC-CP-CA, EBCDIC-CP-NL, EBCDIC-CP-WT, EBCDIC-CP-DK, EBCDIC-CP-NO, EBCDIC-CP-FI, EBCDIC-CP-SE, EBCDIC-CP-IT, EBCDIC-CP-ES, EBCDIC-CP-GB, EBCDIC-CP-FR, EBCDIC-CP-HE, EBCDIC-CP-BE, EBCDIC-CP-CH, EBCDIC-CP-ROECE, EBCDIC-CP-YU, and EBCDIC-CP-IS.

The default encoding is UTF-8. It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) to improve performance as much as 25% over multibyte character sets such as UTF-8.

Output is a pair of C++ source files, .cpp and .h, named after the DTD. Constructors are provided for each class (element) that allow an object to be created in two different ways: initially empty, then adding the children or data after the initial creation, or created with the initial full set of children or initial data. A method is provided for #PCDATA (and Mixed) elements to set the data and, when appropriate, set an element's attributes.

Relevant standards include the W3C recommendation for Extensible Markup Language (XML) 1.0, Document Object Model Level 1 1.0, Namespaces in XML, and the simple API for XML (SAX) 1.0.

Example

The standalone parser may be called as an executable by invoking `bin/xmlcg like xmlcg [flags] <XML document>`. The optional flags are listed in Table 17–1.

Table 17–1 Optional Flags

Flag	Description
`-d`	Specify output directory; default is current directory)
`-e`	Specify default input file encoding

Table 17–1 Optional Flags

Flag	Description
`-h`	Show help instructions

Class XMLClassGenerator

Generates classes based on a DTD.

generate()

Generates classes for the given DTD. Two files are created in the output directory outdir (or in the current directory if outdir is NULL): DTDname.h and DTDname.cpp, both named after the DTD. One class is generated for each defined element in the DTD. Returns `uword` error code, `0` on success

Syntax

```
uword generate(DocumentType *dtd, char *outdir)
```

Parameter	Description
`dtd`	DTD source used to generate the classes
`outdir`	output directory for generated files

Class generated

A generated class is produced for each element defined in the DTD. It has the same name as the element.

Constructors are provided which create an empty element, or make it with an initial set of children or data. Methods are provided to add children or data after construction, and to set attributes. There are two styles of creation: make an empty element, then add the children one at a time, or construct the element with initial data or children. For example, given the element declaration `<!ELEMENT B (#PCDATA | F)*>`, the following constructors will be provided:

```
B(Document *doc);
B(Document *doc, String data);
B(Document *doc, F *theF);
```

The first constructor just makes an empty element with no children. The second initializes it with PCDATA, and the third with a single child node of element F. An element like B which may contain PCDATA is also given a method to add the data post-construction:

```
void addData(Document *doc, String data);
```

The following usages are equivalent:

```
b = new B("data");
```

and

```
b = new B();
b->addData("data");
```

Similarly, the following are also equivalent:

```
f = new F(...);
b = new B(f);
```

and

```
f = new F(...);
b = new B();
b->addNode(f);
```

The presence of modifiers '?' (optional), '*' (zero or more), and '+' (one or more) is ignored when forming the constructors. For example, for the element `<!ELEMENT Sample (A* | (B, (C? | (D, E)*)) | F)+>`, the following constructors are made as if the modifiers were not present:

```
Sample(Document *doc);
Sample(Document *doc, A *theA);
Sample(Document *doc, B *theB, C *theC);
Sample(Document *doc, B *theB, D *theD, E *theE);
Sample(Document *doc, F *theF);
```

If you cannot make the desired final element using one of the forms which take initial children, you must start with the empty element and add nodes as needed with `addNode` as above.

For each attribute for an element, a method is provided to set its value. For example, for the element declaration `<!ELEMENT D (#PCDATA)> ... <!ATTLIST D foo CDATA #REQUIRED>`, the class D will have the method `Attr* setfoo(String value)`.The constructed element is not tested for validity as it is being made. The user to explicitly call the `XMLParser's` validate method on the final element.

Table 17–2 Summary of Methods of generated

Method	Description
Constructor() on page 17-6	Constructs an element that belongs to the document.
addData() on page 17-7	Adds data to the element.
addNode() on page 17-7	Appends a child node to the element.
setattribute() on page 17-7	Sets the element's attribute value.

Constructor()

Constructs an element that belongs to the document. Form one makes the element with no children; use `addData` and addNode as appropriate to populate it. Form two provides initial data or children, depending on the element definition.

Syntax

```
class(Document *doc)
class(Document *doc, ...)
```

Parameter	Description
doc	document to which the element belongs
...	list of arguments; these depends on the element definition

addData()

Adds data to the element. That is, appends to it a PCDATA subnode with the given value. If multiple addData calls are made, the node will have multiple PCDATA subnodes, which should be normalized when construction is finished.

Syntax

```
void addData(Document *doc, String data)
```

Parameter	Description
doc	document to which the element belongs
data	data to be added

addNode()

Appends a child node to the element. No effort is made to validate the resulting element structure at this time; it is the user's responsibility to form the element properly, which may be verified with `XMLParser::validate`.

Syntax

```
void addNode(node thenode)
```

Parameter	Description
node	Node added

setattribute()

Sets the element's attribute value. One method is provided for each attribute, named after the attribute as setattribute. Returns the created attribute.

Syntax

```
Attr* setattribute(String value)
```

Parameter	Description
value	the attribute's value

Index

C

I

P

X