Oracle9*i*

XML API Reference - XDK and Oracle XML DB

Release 2 (9.2)

March 2002

Part No. A96616-01

Oracle9*i* XML API Reference - XDK and Oracle XML DB, Release 2 (9.2)

Part No. A96616-01

Primary Author: Roza Leyderman

Contributing Authors: Joan Gregoire, Shelley Higgins, Diana Lorentz, Jack Melnick, Cathy Shea

Contributors: Nipun Agarwal, Chandramouli Balasubramaniam, Sivasankaran Chandrasekar, Dan Chiba, Mark Drake, Fei Ge, Stanley Guan, Bill Han, Wenyn He, Sam Idicula, Anish Karmarkar, K. Karun, Bhushan Khaladkar, Muralidhar Krishnaprasad, Bruce Lowenthal, Ian Macky, Anjana Manian, Anand Manikutty, Meghna Mehta, Nick Montoya, Steve Muench, Subramanian Muralidhar, Ravi Murthy, Anguel Novoselsky, Visar Nimani, Tomas Saulys, Mark Scardina, Eric Sedlar, Tarvinder Singh, Jinyu Wang, Jim Warner, Simon Wong, Neal Wyse, Tim Yu, Kongyi Zhou

Contents

List of Tables

Send Us Your Comments

Oracle9*i* XML API Reference - XDK and Oracle XML DB, Release 2 (9.2)

Part No. A96616-01

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: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  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 reference describes Oracle XML Developer's Kits (XDK) and Oracle XML DB APIs. It primarily lists the syntax of functions, methods, and procedures associated with these APIs. The Preface contains the following sections:

Audience

This guide is intended for developers building XML applications on Oracle9*i* database.

Organization

This document contains the following chapters:

Chapter 13, "XML Parser for C"

Chapter 14, "XSLT Processor for C"

Chapter 15, "XML Schema Processor for C"

PART III. XDK for C++ Packages

This section describes C++ APIs for XDK.

Chapter 16, "XML Parser for C++"

Chapter 17, "XSLT Processor for C++"

Chapter 18, "XML Schema Processor for C++"

Chapter 19, "XML Class Generator for C++"

PART IV. XDK for PL/SQL

This section describes PL/SQL APIs for XDK.

Chapter 20, "XML SQL Utility (XSU) for PL/SQL"

PART V. XML Database Support: Oracle XML DB for Java

This section describes Java APIs for Oracle XML DB.

Chapter 21, "Java API for XMLType"

Chapter 22, "Oracle XML DB API for Java Beans"

Chapter 23, "Resource APIs for Java/JNDI"

PART VI. XML Database Support: Oracle XML DB for PL/SQL

This section describes PL/SQL APIs for Oracle XML DB.

Chapter 24, "XMLType API for PL/SQL"

Chapter 25, "PL/SQL DOM API for XMLType"

Chapter 26, "PL/SQL Parser API for XMLType"

Chapter 27, "PL/SQL XSLT Processing for XMLType"

Chapter 28, "DBMS_XMLSCHEMA and Catalog Views for PL/SQL"

Chapter 29, "Resource Management and Access Control for PL/SQL"

Chapter 30, "Generating Queries Using DBMS_XMLGEN for PL/SQL"

Chapter 31, "Oracle XML DB Resource View API for PL/SQL"

Chapter 32, "Oracle XML DB Versioning API for PL/SQL"

Chapter 33, "Managing ConText Indexes: DBMS_XDBT in PL/SQL"

PART VII. XML Database Support: SQLX

This section describes SQL APIs for Oracle XML DB.

Chapter 34, "SQLX Functions and Operators"

PART VIII. XML Database Support: Database URI Types

This section describes URI Types APIs for Oracle XML DB.

Chapter 35, "Database URI Types"

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i Database New Features*
- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i XML Case Studies and Applications*

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Database Error Messages*
- *Oracle Text Reference*
- *Oracle Text Application Developer's Guide*
- *Oracle9i Database Concepts*
- *Oracle9i Application Developer's Guide - Fundamentals*
- *Oracle9i Application Developer's Guide - Advanced Queuing*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

Many of the examples in this book use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle9i Sample Schemas* for information on how these schemas were created and how you can use them yourself.

In North America, printed documentation is available for sale in the Oracle Store at

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

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

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/admin/account/membership.html`

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/docs/index.htm`

To access the database documentation search engine directly, please visit

`http://tahiti.oracle.com`

For additional information, see:

- *The Oracle XML Handbook* by XML Core Development Team, Oracle. Oracle Press, 2000.
- *Building Oracle XML Applications* by Steve Muench. O'Reilly & Associates, 2000.
- *XML Bible* byElliotte Rusty Harold. Hungry Minds, Inc., 2001.
- *XML Unleashed* by Morrison et al. SAMS, 1999.
- *Building XML Applications* by St.Laurent and Cerami. McGraw-Hill Professional Publishing, 1999.
- *Building Web Sites with XML* by Michael Floyd. Prentice Hall PTR, 1999.
- *Building Corporate Portals with XML* by Finkelstein, Aiken and Zachman. McGraw-Hill Professional Publishing, 1999.
- *XML in a Nutshell* by Elliotte Rusty Harold and W. Scott Means. O'Reilly & Associates, 2002.
- *XML in a Nutshell : A Desktop Quick Reference* by Elliotte Rusty Harold and W. Scott Means. O'Reilly & Associates, 2001.
- *Learning XML* by Erik T. Ray. O'Reilly & Associates, 2001.
- *XSLT* by Doug Tidwell. O'Reilly & Associates, 2001.
- http://www.xml.com/pub/rg/46
- http://www.xml.org/xmlorg_resources/index.shtml
- http://www.fawcette.com/xmlmag/
- http://www.webmethods.com/
- http://www.xmlwriter.com/
- http://webdevelopersjournal.com/articles/why_xml.html
- http://www.w3schools.com/xml/
- http://www.xml101.com/examples/

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.
`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 italic monospace (fixed-width) font`	Lowercase italic monospace 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]`

...	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS `*`subquery`*`;`
	■ That you can repeat a portion of the code	`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`*

Convention	Meaning	Example

`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;`

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 Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

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.

What’s New with XML Enabled Technology?

This section describes the following new features:

- XML Developer’s Kits Features Introduced in Oracle9i Release 2 (9.2)
- XML Database Features Introduced in Oracle9i Release 2 (9.2)

XML Developer’s Kits Features Introduced in Oracle9*i* Release 2 (9.2)

- **XML Schema Processor for Java**

  Supports the W3C Schema recommendation. Schema Identity-constraint validation no longer needs external DocumentBuilder.

- **XSL Stylesheets**

  Support for threadsafe XSLStylesheet objects.

- **XSQL Servlet**

  New Performance Improvement Option for <xsql:include-owa>.

  <xsql:set-page-param> now supports xpath="Expr" Attribute.

  Simplified inclusion of XML from CLOB and VARCHAR2 Columns.

  New <xsql:include-posted-xml> action handler to include posted XML.

  Support for the Apache FOP 0.19 release.

  Supports Immediately Read Values Set as Cookies.

  Supports Setting Multiple Parameter Values with a Single SQL Statement.

- **Class Generator for Java**

  Data Binding Feature is added to the DTD Class Generator.

  An XML instance document could be given as input to load the instance data to the generated classes.

- **XDK for Java**

  XSU support for SAX 2.0 and generating the XML Schema of a SQL Query.

  DOM level compression support.

  Oracle SOAP APIs added.

  SAX2 Extension support in the Java XML Parser.

  JAXP 1.1 support is now provided by the XDK for Java.

  Oracle TransX Utility aids loading data and text.

  XML Schema Processor for Java supports both LAX Mode and STRICT Mode.

  XML Compression now supported in the Java XML Parser.

- **XDK for C**

  Released on Linux.

- **XDK for C++**

  Released on Linux.

- **XDK for JavaBeans**

  New XMLDiff Bean.

  Internal DTD support is added to the SourceViewer Bean.

- **OTN**

  New XDK Live demo is online at:

  `http://otn.oracle.com/tech/xml/xdk_sample/xdkdemo_faq.html`

  `http://otn.oracle.com/tech/xml/xdk_sample/xdkdemo_xsql.html`

  New XDK technical Paper for "Building Server-Side XML Schema Validation" is online at `http://otn.oracle.com/tech/xml/xdk_sample/xdksample_093001i.html`

XML Database Features Introduced in Oracle9*i* Release 2 (9.2)

- **XMLType Tables**

  Datatype `XMLType` can now be used to create tables of `XMLType`.

- **XMLType Constructors**

  Additional `XMLType` constructors have been added.

- **W3C XML Schema Support**

  Construct an `XMLType` object based on an XML schema and have it be continuously validated.

  Create XML schema-based `XMLType` tables.

  Register annotated XML schema using the `DBMS_XMLSCHEMA` package, to share storage and type definitions.

  Pre-parse incoming XML documents and automatically direct them to default storage tables.

Automatically validate XML documents or instances against W3C XML Schema when the XML documents or instances are added to Oracle XML DB.

Explicitly validate XML documents and instances against XML schema using `XMLIsSchemaValid()` method on `XMLType`.

Use datatype -aware extraction of part of an XML document using the `extractValue` operator.

- **SQLX Functions and Oracle Extensions**

  Generating XML from existing relational and object relational tables. This is based on ISO-ANSI Working Draft for XML-Related Specifications (SQL/XML) [ISO/IEC 9075 Part 14 and ANSI] which defines ways in which the database language SQL can be used in conjunction with XML.

- **W3C XPath Support for Extraction, Condition Checks, and Updates**

  `extract()`, `existsNode()`, and `extractValue()` support namespace-based operation.

  `extract()`, `existsNode()`, and `extractValue()` support the full XPath function set, including axis operators.

  `updateXML()` replaces part of the `XMLType` DOM by using XPath as a locator.

- **ToObject Method**

  `ToObject` method allows the caller to convert an `XMLType` object to a PL/SQL object type.

- **XMLType Views**

  This release supports `XMLType`-based views. These enable you to view any data in the database as XML. `XMLType` views can be XML schema-based or non-XML schema-based.

- **W3C XSLT Support**

  `XMLTransform()` allows database-based transformation of cached or on disk XML documents.

- **JDBC Support for XMLType**

  Oracle XML DB allows database clients to bind and define `XMLType`. JDBC support includes a function-rich `XMLType` class that allows for native (for thick JDBC) XML functionality support.

- **C-Based PL/SQL DOM, Parser, and XSLT APIs**

  This release includes native PL/SQL DOM, Parser, and XSLT APIs integrated in the database code. These PL/SQL APIs are compatible with the Java-based PLSQL APIs shipped as part of XDK for PL/SQL with Oracle9*i* Release 1 (9.0.1) and higher..

- **Oracle XML DB: Repository**

  Viewing the database and its content as a file system containing resources, typically referred to as files and folders.

  Access to resources through SQL and Java API, using path names.

  Access to resources through built-in native Protocol Servers for FTP, HTTP, and WebDAV.

  ACL-based security for Oracle XML DB resources.

- **Oracle XML DB Resource API (PL/SQL): DBMS_XDB**

  `DBMS_XDB` package provides methods to access and manipulate Oracle XML DB resources.

- **Oracle XML DB Resource API (JNDI)**

  Locate resources and manage collections. Works only inside the database server on the JServer platform.

- **Oracle XML DB Resource View API (SQL)**

  `ResourceView` is a public `XMLType` view can be used to perform queries against all resources in a database instance, based on path names.

- **Oracle XML DB Versioning: DBMS_XDB_VERSION**

  `DBMS_XDB_VERSION` package provides methods to version Oracle XML DB resources.

- **Oracle XML DB ACL Security**

  Methods that implement ACL-based security have been developed as a part of `DBMS_XDB` package. They enable the creation of high-performance access control lists for any `XMLType` object.

- **Oracle XML DB Protocol Servers**

  The Protocol Servers provide access to any foldered `XMLType` row through FTP, HTTP, and WebDAV.

- **XDBURIType**

  XDBURIType, a subtype of URIType, represents a path name within Oracle XML DB.

- **Oracle Enterprise Manager**

  Oracle Enterprise Manager provides a graphical interface to manage, administer, and configure Oracle XML DB.

- **Oracle Text Enhancements**

  Columns of type XMLType can now be indexed natively in Oracle9*i* database using Oracle Text.

  CONTAINS() is a new function for use as ora:contains in an XPath query and as part of the existsNode() function.

  CTXXPATH is a new index type for use with existsNode() to speedup the performance of XPath searching.

- **Oracle Advanced Queuing (AQ) Support**

  IDAP has been added to facilitate use of AQ over the Internet; the AQ XML servlet can access Oracle9*i* AQ using HTTP and SOAP.

  IDAP defines the XML message structure used in the body of the SOAP request.

  XMLType can be used as the AQ payload type directly, instead of having to embed XMLType as an attribute in an Oracle object type.

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

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

DefaultXMLDocumentHandler Class

Description of DefaultXMLDocumentHandler

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 of DefaultXMLDocumentHandler

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

`oracle.xml.parser.v2.DefaultXMLDocumentHandler`

Implemented Interfaces of DefaultXMLDocumentHandler

`XMLDocumentHandler`

Methods of 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 1-4	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 1-5	End the scope of a prefix-URI mapping.
getHandler() on page 1-5	Get the next pipe-line node handler.
setDoctype() on page 1-6	Receive notification of DTD. Sets the DTD.
setError() on page 1-6	Receive notification of a XMLError handler.
setHandler() on page 1-6	Receive notification of a next pipe-line node handler.
setTextDecl() on page 1-7	Receives notification of a Text XML Declaration.
setXMLDecl() on page 1-7	Receives notification of an XML Declaration.

Table 1–1 Summary of Methods of DefaultXMLDocumentHandler (Cont.)

Method	Description
setXMLSchema() on page 1-8	Receives notification of a XMLSchema object.
skippedEntity() on page 8	Receives notification of a skipped entity.
startElement() on page 1-9	Receives notification of the beginning of an element.
startPrefixMapping() on page 1-10	Begins the scope of a prefix-URI Namespace mapping.

DefaultXMLDocumentHandler()

Description

Constructs a default document.

Syntax

```
public  DefaultXMLDocumentHandler();
```

cDATASection()

Description

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()

Description

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()

Description

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()

Description

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.

Syntax	Description
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()

Description
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()

Description
Returns the next pipe-line node handler node.

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

setDoctype()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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

Syntax

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

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

DocumentBuilder Class

Syntax of DocumentBuilder

```
public class DocumentBuilder
```

`oracle.xml.parser.v2.DocumentBuilder`

Description of DocumentBuilder

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

Methods of DocumentBuilder

Table 1–2 Summary of Methods of DocumentBuilder

Method	Description
DocumentBuilder() on page 1-12	Creates a document builder that can be used as XMLDocumentHandler.
attributeDecl() on page 1-13	Reports an attribute type declaration.
cDATASection() on page 1-13	Receives notification of CDATA Section data inside an element.
characters() on page 1-14	Receives notification of character data inside an element.
comment() on page 1-14	Receives notification of a comment.
elementDecl() on page 1-15	Reports an element type declaration.
endCDATA() on page 1-16	Reports the end of a CDATA section.
endDoctype() on page 1-16	Receives notification of end of the DTD.
endDocument() on page 1-16	Receives notification of the end of the document.
endDTD() on page 1-16	Reports the end of DTD declarations.
endElement() on page 1-17	Receives notification of the end of an element.
endEntity() on page 1-17	Reports the end of an entity.
externalEntityDecl() on page 1-18	Reports a parsed external entity declaration.
getCurrentNode() on page 1-18	Returns the current node being build.

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

Method	Description
getDocument() on page 1-19	Gets the document being build
ignorableWhitespace() on page 1-19	Receives notification of ignorable whitespace in element content.
internalEntityDecl() on page 1-19	Reports an internal entity declaration
processingInstruction() on page 1-20	Receives notification of a processing instruction.
retainCDATASection() on page 1-20	Sets a flag to retain CDATA sections
setDebugMode() on page 1-21	Sets a flag to turn on debug information in the document
setDoctype() on page 1-21	Receives notification of DTD Sets the DTD
setDocumentLocator() on page 1-21	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 1-22	Sets an optional NodeFactory to be used for creating custom DOM trees
setTextDecl() on page 1-22	Receives notification of a Text XML Declaration. The Parser will invoke this method once for each text XML Decl.
setXMLDecl() on page 1-23	Receives notification of a XML Declaration. The Parser will invoke this method once for XML Decl.
startCDATA() on page 1-23	Reports the start of a CDATA section.
startDocument() on page 1-23	Receives notification of the beginning of the document.
startDTD() on page 1-24	Reports the start of DTD declarations, if any.
startElement() on page 1-24	Receives notification of the beginning of an element.
startEntity() on page 1-25	Reports the beginning of some internal and external XML entities. All start/indemnity events must be properly nested.

DocumentBuilder()

Description

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

Syntax

```
public  DocumentBuilder();
```

attributeDecl()

Description

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()

Description

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()

Description

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()

Description

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.
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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Returns the current XMLNode being build.

Syntax

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

getDocument()

Description

Returns the document being build.

Syntax

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

ignorableWhitespace()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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

Syntax

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

Parameter	Description
f	NodeFactory/

setTextDecl()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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.
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()

Description

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

Syntax of DOMParser

```
public class DOMParser
```

oracle.xml.parser.v2.DOMParser

Description of DOMParser

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.

Fields of 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

Methods of DOMParser

Table 1–4 Summary of Methods of DOMParser

Method	Description
DOMParser() on page 1-27	Creates a new parser object.

Table 1–4 Summary of Methods of DOMParser

Method	Description
getAttribute() on page 1-27	Returns specific attributes of the underlying implementation.
getDoctype() on page 1-28	Gets the DTD.
getDocument() on page 1-28	Gets the document.
parseDTD() on page 1-28	Parses the XML External DTD from given input source.
reset() on page 1-29	Resets the parser state
retainCDATASection() on page 1-29	Switches to determine whether to retain CDATA sections
setAttribute() on page 1-30	Sets specific attributes on the underlying implementation.
setDebugMode() on page 1-30	Sets a flag to turn on debug information in the document
setErrorStream() on page 1-31	Creates an output stream for errors and warnings.
setNodeFactory() on page 1-31	Sets the node factory.
showWarnings() on page 1-32	Switches to determine whether to print warnings

DOMParser()

Description

Creates a new parser object.

Syntax;

```
public  DOMParser();
```

getAttribute()

Description

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()

Description
Returns the DTD.

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

getDocument()

Description
Returns the document being parsed.

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

parseDTD()

Description
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.

Syntax	Description
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()

Description
Resets the parser state.

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

retainCDATASection()

Description
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()

Description

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	he value of the attribute.

setDebugMode()

Description

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()

Description

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()

Description

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()

Description

Switches to determine whether to print warnings.

Syntax

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

Parameter	Description
flag	Determines whether warnings should be shown.

NodeFactory Class

Description of NodeFactory

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 of NodeFactory

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

java.lang.Object
  |
  +--oracle.xml.parser.v2.NodeFactory
```

Implemented Interfaces of NodeFactory

`java.io.Serializable`

Methods of NodeFactory

Table 1–5 Summary of Methods of NodeFactory

Method	Description
NodeFactory() on page 1-34	Class constructor.
createAttribute() on page 1-34	Creates and returns an attribute node with the specified tag, and text.
createCDATASection() on page 1-35	Creates and returns a CDATA node with the specified text.
createComment() on page 1-35	Creates and returns a comment node with the specified text.
createDocument() on page 1-36	Creates and returns a document node. This method cannot return a null pointer.
createDocumentFragment() on page 1-36	Creates and returns a document fragment node with the specified tag.

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

Method	Description
createElement() on page 1-36	Creates and returns an Element node with the specified tag.
createElementNS() on page 1-37	Creates and returns an Element node with the specified local name, prefix, namespaceURI
createEntityReference() on page 1-37	Creates and returns an entity reference node with the specified tag.
createProcessingInstruction() on page 1-38	Creates and returns a PI node with the specified tag, and text.
createTextNode() on page 1-38	Creates and returns s a text node with the specified text.

NodeFactory()

Description

Class constructor.

Syntax

```
public  NodeFactory();
```

createAttribute()

Description

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.

Syntax	Description
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()

Description

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()

Description

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()

Description

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

Syntax

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

createDocumentFragment()

Description

Creates a document fragment node with the specified tag.

Syntax

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

Returns

The created document fragment node.

createElement()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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

Description of oraxml

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.

Syntax of oraxml

```
public class oraxml extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.parser.v2.oraxml
```

Methods of oraxml

oraxml()

Syntax

```
public oraxml();
```

main()

syntax

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

SAXAttrList Class

Description of SAXAttrList

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 of SAXAttrList

```
public class SAXAttrList
```

`oracle.xml.parser.v2.SAXAttrList`

Methods of SAXAttrList

Table 1–7 Summary of Methods of SAXAttrList

Method	Description
SAXAttrList() on page 1-41	Class constructor.
addAttr() on page 1-41	Adds an attribute to the parent element node.
getExpandedName() on page 1-42	Returns the expanded name for an attribute in the list (by position).
getIndex() on page 1-43	Returns the index of an attribute.
getLength() on page 1-43	Returns the number of attributes in the list.
getLocalName() on page 1-43	Returns an attribute's local name by index.

Table 1–7 Summary of Methods of SAXAttrList (Cont.)

Method	Description
getPrefix() on page 1-44	Returns the namesp.ace prefix for an attribute in the list by position.
getQName() on page 1-44	Returns an attribute's qualified name by index.
getType() on page 1-45	Returns an attribute's type.
getURI() on page 1-45	Returns an attribute's Namespace URI by index
getValue() on page 1-46	Returns the attribute's value as a string,
reset() on page 1-46	Resets the SAXAttrList.

SAXAttrList()

Description

Class constructor.

Syntax

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

addAttr()

Description

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.

Syntax	Description
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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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).

Parameters

`i` - The index of the attribute in the list.

getQName()

Description

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()

Description

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).
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()

Description

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()

Description

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()

Description

Resets the SAXAttrList.

Syntax

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

SAXParser Class

Description of SAXParser

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 of SAXParser

```
public class SAXParser
```

`oracle.xml.parser.v2.SAXParser`

Methods of SAXParser

Table 1–8 Summary of Methods of SAXParser

Method	Description
SAXParser() on page 1-48	Creates a new parser object.
getContentHandler() on page 1-48	Returns the current content handler.
getDTDHandler() on page 1-48	Returns the current DTD handler.
getFeature() on page 1-49	Returns the current state of the feature (true or false).

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

Method	Description
getProperty() on page 1-49	Returns the value of a property.
setContentHandler() on page 1-50	Registers a content event handler.
setDTDHandler() on page 1-50	Registers a DTD event handler.
setFeature() on page 1-51	Set the state of a feature.
setProperty() on page 1-52	Sets the value of a property.

SAXParser()

Description

Creates a new parser object.

Syntax

```
public  SAXParser()
```

getContentHandler()

Description

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

Syntax

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

getDTDHandler()

Description

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

Syntax

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

getFeature()

Description

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.

The feature http://www.xml.org/sax/features/validation due to its binary input value 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()

Description

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()

Description

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()

Description

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. Throws `java.lang.NullPointerException` if the handler argument is null.

Syntax

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

Parameter	Description
handler	The DTD handler.

setFeature()

Description

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. The feature "http:://www.xml.org/sax/features/validation" due to its binary input value 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()

Description

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

Description of XMLParseException

Indicates that a parsing exception occurred while processing an XML document

Syntax of XMLParseException

```
public class XMLParseException
```

`oracle.xml.parser.v2.XMLParseException`

Fields of XMLParseException

Table 1–9 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

Methods of XMLParseException

Table 1–10 Summary of Methods of XMLParseException

Method	Description
XMLParseException() on page 1-54	Constructor for the XMLParse Exception class.
formatErrorMessage() on page 1-54	Formats error message at specified index.
getColumnNumber() on page 1-55	Returns column number of error at specified index.
getException() on page 1-55	Returns the exception that occurred in error at specified index.
getLineNumber() on page 1-55	Returns the line number of error at specified index.
getMessage() on page 1-56	Returns the error message at specified index.
getMessageType() on page 1-56	Returns the type of message at specified index.
getNumMessages() on page 1-56	Returns the total number of errors / warnings found during parsing.

Table 1–10 Summary of Methods of XMLParseException (Cont.)

Method	Description
getPublicId() on page 1-57	Returns the public id of input when error at specific index occurred.
getSystemId() on page 1-57	Returns the system ID of input when error at specific index occurred.

XMLParseException()

Description

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()

Description

Returns the error message at specified index.

Syntax

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

Parameter	Description
i	Index.

getColumnNumber()

Description

Returns the column number of error at specified index.

Syntax

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

Parameter	Description
i	Index.

getException()

Description

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()

Description

Returns the line number of error at specified index

Syntax

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

Parameter	Description
i	Index.

getMessage()

Description

Returns the error message at specified index

Syntax

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

Parameter	Description
i	Index.

getMessageType()

Description

Returns the type of the error message at specified index.

Syntax

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

Parameter	Description
i	Index.

getNumMessages()

Description

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

Syntax

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

getPublicId()

Description

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

Syntax

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

Parameter	Description
i	Index

getSystemId()

Description

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

Syntax

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

Parameter	Description
i	Index

XMLParser Class

Description of XMLParser

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).

Syntax of XMLParse

```
public abstract class XMLParser
```

`oracle.xml.parser.v2.XMLParser`

Fields of XMLParser

Table 1–11 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 1–12 Summary of Methods of XMLParser

Method	Description
getAttribute() on page 1-60	Retrieves value of attribute of the implementation.
getBaseURL() on page 1-60	Returns the base URL.
getEntityResolver() on page 1-60	Returns the entity resolver.
getErrorHandler() on page 1-61	Returns the error handler.
getReleaseVersion() on page 1-61	Returns the release version.
getValidationModeValue() on page 1-61	Returns the validation mode value.
getXMLProperty() on page 1-61	Returns the value of a property.
isXMLPropertyReadOnly() on page 1-62	Returns TRUE if a given property is read-only.
isXMLPropertySupported() on page 1-62	Returns TRUE if a given property is supported.
parse() on page 1-63	Parses the XML.
reset() on page 1-63	Resets the parser state.
setAttribute() on page 1-64	Sets specific attributes on the underlying implementation.
setBaseURL() on page 1-64	Sets the base URL for loading external entities and DTDs.
setDoctype() on page 1-64	Sets the DTD.
setEntityResolver() on page 1-65	Sets the entity resolver.
setErrorHandler() on page 1-65	Registers an error event handler.
setLocale() on page 1-66	Sets the locale for error reporting.
setPreserveWhitespace() on page 1-66	Sets the white space preserving mode.
setValidationMode() on page 1-67	Sets the validation mode.
setXMLProperty() on page 1-67	Sets and returns an XMLProperty.
setXMLSchema() on page 1-67	Sets an XML Schema for validating the instance document.

getAttribute()

Description

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()

Description

Returns the base URL.

Syntax

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

getEntityResolver()

Description

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()

Description

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

Syntax;

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

getReleaseVersion()

Description

Returns the release version of the Oracle XML Parser.

Syntax

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

getValidationModeValue()

Description

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()

Description

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()

Description

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()

Description

Returns TRUE if a given property is supported.

Syntax

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

Parameter	Description
name	Name of the property.

parse()

Description

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.
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()

Description

Resets the parser state.

Syntax

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

setAttribute()

Description

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()

Description

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()

Description

Sets the DTD.

Syntax

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

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

setEntityResolver()

Description

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()

Description

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()

Description

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()

Description

Sets the white space preserving mode.

Syntax

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

Parameter	Description
flag	The preserving mode.

setValidationMode()

Description

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()

Description

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()

Description

Sets an XMLSchema for validating the instance document.

Syntax

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

Parameter	Description
schema	The XMLSchema object

XMLToken Class

Description of XMLToken

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 of XMLToken

```
public interface XMLToken
```

Fields of XMLToken of XMLToken

Table 1–13 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 ::= ']]>'

Table 1–13 Fields of XMLToken (Cont.)

Field	**Syntax**	**Description**
CharData	public static final int CharData	CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
Comment	public static final int Comment	Comment ::= '<' '!' '--' ((Char - '-') | ('-' (Char - '-')))* '-->'
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'))

Table 1–13 Fields of XMLToken (Cont.)

Field	Syntax	Description
Reference	public static final int Reference	Reference ::= EntityRef | CharRef | PEReference EntityRef ::= '&' Name ';' PEReference ::= '%' Name ';' CharRef ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';
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? '?' '>'

Methods of XMLToken

token()

Description

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

Description of XMLTokenizer

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

Syntax of XMLTokenizer

```
public class XMLTokenizer
```

oracle.xml.parser.v2.XMLTokenizer

Methods of XMLTokenizer

Table 1–14 Summary of Methods of XMLTokenizer

Method	Description
XMLTokenizer() on page 1-72	Creates a new Tokenizer object.
parseDocument() on page 1-73	Parses the document.
setErrorHandler() on page 1-73	Registers a new error event handler.
setErrorStream() on page 1-73	Registers a output stream for errors.
setToken() on page 1-74	Registers a new token for XML tokenizer.
setTokenHandler() on page 1-74	Registers a new XML tokenizer event handler.
tokenize() on page 1-74	Tokenizes the XML.

XMLTokenizer()

Description

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()

Description

Parses the document.

Syntax

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

setErrorHandler()

Description

Registers a new error event handler. This replaces any previous setting for error handling.

Syntax

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

Parameter	Description
handler	The handler being registered.

setErrorStream()

Description

Registers a output stream for errors.

Syntax

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

Parameter	Description
out	Error stream being registered.

setToken()

Description

Registers a new token for XML tokenizer.

Syntax

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

Parameter	Description
token	XMLToken being set.

setTokenHandler()

Description

Registers a new XML tokenizer event handler.

Syntax

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

Parameter	Description
handler	XMLToken being registered.

tokenize()

Description

Tokenizes 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 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

Description of NSName

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

Syntax of NSName

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

Methods of NSName

Table 1–15   Summary of Methods of NSName

Method	Description
getExpandedName() on page 1-76	Returns the fully resolved name for this name.
getLocalName() on page 1-76	Returns the local name for this name.
getNamespace() on page 1-77	Returns the resolved Namespace for this name.
getPrefix() on page 1-77	Returns the prefix for this name.
getQualifiedName() on page 1-77	Returns the qualified name.

getExpandedName()

Description

Returns the fully resolved name for this name.

Syntax

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

getLocalName()

Description

Returns the local name for this name.

Syntax

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

getNamespace()

Description

Returns the resolved Namespace for this name.

Syntax

```
public String getNamespace()
```

getPrefix()

Description

Returns the prefix for this name.

Syntax

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

getQualifiedName()

Description

Returns the qualified name.

Syntax

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

XMLError Class

Description of XMLError

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

Syntax of XMLError

```
public class XMLError
```

`oracle.xml.util.XMLError`

Fields of XMLError

Table 1–16 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

Methods of XMLError

Table 1–17 Summary of Methods of XMLError

Method	Description
XMLError() on page 1-80	Default constructor for XMLError.
error() on page 1-80	Adds a new error to the vector.
error0() on page 1-81	Adds a new error to the vector, with no additional parameters.

Table 1–17 Summary of Methods of XMLError (Cont.)

Method	Description
error1() on page 1-81	Adds a new error to the vector, with 1 additional parameter.
error2() on page 1-82	Adds a new error to the vector, with 2 additional parameters.
error3() on page 1-82	Adds a new error to the vector, with 3 additional parameters.
flushErrorStream() on page 1-83	Flush all the error to the output stream output stream defaults or to error handler
formatErrorMesg() on page 1-83	Formats and error message.
getColumnNumber() on page 1-83	Returns the column number of error at specified index
getException() on page 1-84	Returns the exception (if exists) that occured in error at specified index
getFirstError() on page 1-84	Returns the first error.
getLineNumber() on page 1-84	Returns the line number of error at specified index.
getLocator() on page 1-85	Returns the registered locator.
getMessage() on page 1-85	Returns the error message at specified index
getMessage0() on page 1-85	Returns error message with no parameters.
getMessage1() on page 1-86	Returns error message with 1 parameter.
getMessage2() on page 1-86	Returns error message with 2 parameters.
getMessage3() on page 1-87	Returns error message with 3 parameters.
getMessage4() on page 1-87	Returns error message with 4 parameters.
getMessage5() on page 1-88	Returns error message with 5 parameters.
getMessageType() on page 1-88	Returns the type of the error message at specified index.
getNumMessages() on page 1-89	Returns the total number of errors/warnings found during parsing.
getPublicId() on page 1-89	Returns the public ID of input when error at specified index occured.
getSystemId() on page 1-89	Returns the system ID of input when error at specified index occured.
printErrorListener() on page 1-90	Flushes all the JAXP 1.1 errors to the ErrorListener If no ErrorListener was set, default to System.err.
reset() on page 1-90	Resets the error class.

Table 1–17 Summary of Methods of XMLError (Cont.)

Method	Description
setErrorStream() on page 1-90	Registers an output stream.
setException() on page 1-91	Registers an exception.
setLocale() on page 1-91	Registers a locale.
setLocator() on page 1-91	Registers a locator.
showWarnings() on page 1-92	Turns reporting warning on / off.

XMLError()

Description

Default constructor for XMLError.

Syntax

```
public  XMLError();
```

error()

Description

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.

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

error0()

Description

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()

Description

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()

Description

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()

Description

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.

Parameter	Description
p2	Parameter 2.
p3	Parameter 3.

flushErrorStream()

Description

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

Syntax

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

formatErrorMesg()

Description

Formats an error message.

Syntax

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

Parameter	Description
i	Index.

getColumnNumber()

Description

Returns the column number of error at specified index.

Syntax

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

Parameter	Description
i	Index.

getException()

Description

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()

Description

Returns the first error.

Syntax

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

getLineNumber()

Description

Returns the line number of error at specified index.

Syntax

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

Parameter	Description
i	Index.

getLocator()

Description

Returns the registered locator

Syntax

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

getMessage()

Description

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()

Description

Returns error message with no parameters.

Syntax

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

Parameter	Description
errId	The error id.

getMessage1()

Description

Returns error message with 1 argument.

Syntax

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

Parameter	Description
errId	The error id.
a1	Parameter 1.

getMessage2()

Description

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()

Description

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()

Description

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.

Parameter	Description
a2	Parameter 2.
a3	Parameter 3.
a4	Parameter 4.

getMessage5()

Description

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()

Description

Returns the type of the error message type at specified index

Syntax

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

Parameter	Description
i	Index

getNumMessages()

Description

Returns the total number of errors / warnings found during parsing

Syntax

```
public int getNumMessages()
```

getPublicId()

Description

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

Syntax

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

Parameter	Description
i	Index.

getSystemId()

Description

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

Syntax

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

Parameter	Description
i	Index.

printErrorListener()

Description

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

Syntax

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

reset()

Description

Resets the error class.

Syntax

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

setErrorStream()

Description

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()

Description
Registers an exception.

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

Parameter	Description
exp	Last exception which occured.

setLocale()

Description
Registers a locale for error reporting.

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

Parameter	Description
locale	The locale for error reporting.

setLocator()

Description
Register a locator

Syntax

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

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

showWarnings()

Description

Turns reporting warning on / off.

Syntax

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

Parameter	Description
flag	Controls reporting of warnings.

XMLException Class

Description of XMLException

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

Syntax of XMLException

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

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.xml.util.XMLException
```

Implemented Interfaces of XMLException

```
java.io.Serializable
```

Fields of XMLException

Table 1–18 Fields of XMLException

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

Methods of XMLException

Table 1–19 Summary of Methods of oracle.xml.util.XMLException

Method	Description
XMLException() on page 1-94	Generates an XMLException.
formatErrorMessage() on page 1-95	Returns the error message at specified index.
getColumnNumber() on page 1-96	Returns the column number of error at specified index.

Table 1–19 Summary of Methods of oracle.xml.util.XMLException (Cont.)

Method	Description
getException() on page 1-96	Returns the exception (if exists) that occured in error at specified index.
getLineNumber() on page 1-96	Returns the line number of error at specified index.
getMessage() on page 1-97	Returns the error message at specified index.
getMessageType() on page 1-97	Returns the type of the error message at specified index.
getNumMessages() on page 1-97	Returns the total number of errors / warnings found during parsing.
getPublicId() on page 1-98	Returns the public ID of input when error at specified index occured.
getSystemId() on page 1-98	Returns the system ID of input when error at specified index occured.
getXMLError() on page 1-98	Returns XMLError object inside XMLException.
printStackTrace() on page 1-99	Prints this `Throwable` and its backtrace.
setException() on page 1-99	Sets the underlying exception, if exists.
toString() on page 1-99	Returns any embedded exception.

XMLException()

Description

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.

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

formatErrorMessage()

Description

Returns the error message at specified index.

Syntax

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

Parameter	Description
i	Index.

getColumnNumber()

Description

Returns the column number of error at specified index.

Syntax

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

Parameter	Description
i	Index.

getException()

Description

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()

Description

Returns the line number of error at specified index

Syntax

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

Parameter	Description
i	Index.

getMessage()

Description

Returns the error message at specified index

Syntax

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

Parameter	Description
i	Index.

getMessageType()

Description

Get the type of the error message at specified index

Syntax

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

Parameter	Description
i	Index.

getNumMessages()

Description

Returns the total number of errors/warnings found during parsing

Syntax

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

getPublicId()

Description

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

Syntax

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

Parameter	Description
i	Index.

getSystemId()

Description

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

Syntax

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

Parameter	Description
i	Index.

getXMLError()

Description

Get XMLError object inside XMLException.

Syntax

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

printStackTrace()

Description

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()

Description

Sets the underlying exception, if it exists.

Syntax

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

Parameter	Description
ex	The exception

toString()

Description

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:

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

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

NSResolver Interface

Syntax of NSResolver

```
public interface NSResolver
```

Description of NSResolver

This interface provides support for resolving Namespaces.

Implementing Classes of NSResolver

`XMLElement`

Methods of NSResolver

resolveNamespacePrefix()

Description

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

Description of PrintDriver

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

Syntax of PrintDriver

```
public interface PrintDriver
```

Implementing Classes of PrintDriver

`XMLPrintDriver`

Methods of 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 2-6	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 2-7	Prints a DTD.
printDocument() on page 2-7	Prints an XMLDocument.
printDocumentFragment() on page 2-7	Prints an empty XMLDocumentFragment object.
printElement() on page 2-8	Prints an XMLElement.
printEntityReference() on page 2-8	Prints an XMLEntityReference node.
printProcessingInstruction() on page 2-8	Prints an XMLPI node.
printTextNode() on page 2-9	Prints an XMLText node.
setEncoding() on page 2-9	Sets the encoding of the print driver.

close()

Description

Closes the output stream or print writer

Syntax

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

flush()

Description

Flushes the output stream or print writer

Syntax

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

printAttribute()

Description

Prints an `XMLAttr` node

Syntax

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

Parameter	Description
attr	The XMLAttr node.

printAttributeNodes()

Description

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()

Description
Prints an `XMLCDATA` node.

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

Parameter	Description
cdata	The XMLCDATA node.

printChildNodes()

Description
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()

Description
Prints an `XMLComment` node.

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

Parameter	Description
comment	The comment node.

printDoctype()

Description

Prints a `DTD`.

Syntax

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

Parameter	Description
dtd	The DTD to be printed.

printDocument()

Description

Prints an `XMLDocument`.

Syntax

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

Parameter	Description
doc	The document to be printed.

printDocumentFragment()

Description

Prints an empty `XMLDocumentFragment` object.

Syntax

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

Parameter	Description
dfrag	The document fragment to be printed.

printElement()

Description

Prints an `XMLElement`.

Syntax

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

Parameter	Description
elem	The element to be printed.

printEntityReference()

Description

Prints an `XMLEntityReference` node.

Syntax

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

Parameter	Description
eref	The XMLEntityReference node to be printed.

printProcessingInstruction()

Description

Prints an `XMLPI` node.

Syntax

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

Parameter	Description
pi	The XMLPI node to be printed.

printTextNode()

Description

Prints an `XMLText` node.

Syntax

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

Parameter	Description
text	The text node.

setEncoding()

Description

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

Description of AttrDecl

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

Syntax of AttrDecl

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

oracle.xml.parser.v2.AttrDecl

Implemented Interfaces of AttrDecl

`java.io.Externalizable, java.io.Serializable`

Fields of AttrDecl

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

Methods of AttrDecl

Table 2–3 Summary of Methods of AttrDecl

Methods	Description
AttrDecl() on page 2-11	Default constructor.
getAttrPresence() on page 2-11	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 2-13	Returns a code representing the type of the underlying object.
readExternal() on page 2-13	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 2-14	Saves the state of the object by creating a binary compressed stream with information about this object.

AttrDecl()

Description

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

getAttrPresence()

Description

Returns attribute presence.

Syntax

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

getAttrType()

Description

Returns attribute type.

Syntax

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

getDefaultValue()

Description

Returns attribute default value.

Syntax

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

getEnumerationValues()

Description

Returns attribute values as an `Enumeration`.

Syntax

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

getNodeName()

Description

Returns the name of the Attr Decl.

Syntax

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

getNodeType()

Description

Returns a code representing the type of the underlying object.

Syntax

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

readExternal()

Description

Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. 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()

Description

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()

Description

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.

DTD Class

Description of DTD

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

Syntax of DTD

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

oracle.xml.parser.v2.DTD

Implemented Interfaces of DTD

```
java.io.Externalizable, java.io.Serializable
```

Methods of DTD

Table 2–4 Summary of Methods of DTD

Method	Description
DTD() on page 2-16	Default constructor.
findElementDecl() on page 2-17	Finds and returns an element declaration for the given tag name.
findEntity() on page 2-17	Finds and returns a named entity in the DTD; returns null if it is not found.
findNotation() on page 2-17	Retrieves the named notation from the DTD; returns null if it is not found.
getChildNodes() on page 2-18	Returns a NodeList that contains all children of this node.
getElementDecls() on page 2-18	Returns a NamedNodeMap containing the element declarations in the DTD.
getEntities() on page 2-18	Returns a NamedNodeMap containing the general entities, both external and internal, declared in the DTD.
getInternalSubset() on page 2-19	Returns the internal subset of the DTD.
getName() on page 2-19	Returns the name of the DTD, or the name immediately following the DOCTYPE keyword.

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

Method	Description
getNodeName() on page 2-19	Returns the name of the DTD, or the name immediately following the DOCTYPE keyword.
getNodeType() on page 2-19	Returns a code representing the type of the underlying object.
getNotations() on page 2-20	Returns a NamedNodeMap containing the notations declared in the DTD.
getOwnerImplementation() on page 2-20	Returns the owner of the DTD implementation.
getPublicId() on page 2-20	Returns the public identifier associated with the DTD, if specified.
getRootTag() on page 2-20	Returns the root tag for the DTD.
getSystemId() on page 2-21	Returns the system identifier associated with the DTD, if specified. If the system identifier was not specified, this is null.
hasChildNodes() on page 2-21	Determines whether a node has any children; return false always, as DTD cannot have any overrides method in XMLNode.
normalize() on page 2-21	Normalizes the DTD.
printExternalDTD() on page 2-21	Writes the contents of this document to the given output.
readExternal() on page 2-22	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
setRootTag() on page 2-22	Sets the root tag for the DTD.
writeExternal() on page 2-23	Saves the state of the object by creating a binary compressed stream with information about this object.

DTD()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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 The DOM Level 1 does not support editing ElementDecl, therefore `ElementDecl` cannot be altered in any way.

Syntax

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

getEntities()

Description

Returns a `NamedNodeMap` containing the general entities, both external and internal, declared in the DTD. Duplicates are discarded. For example in:<!DOCTYPE ex SYSTEM "ex.dtd" [<!ENTITY foo "foo"> <!ENTITY bar "bar"> <!ENTITY % baz "baz">]> <ex/> the interface provides access to `foo` and `bar` but not `baz`. Every node in this map also implements the `Entity` interface. The DOM Level 1 does not support editing entities, therefore `entities` cannot be altered in any way.

Syntax

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

getInternalSubset()

Description

Returns the internal subset of the DTD.

Syntax

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

getName()

Description

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

Syntax

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

getNodeName()

Description

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

Syntax

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

getNodeType()

Description

Returns a code representing the type of the underlying object.

Syntax

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

getNotations()

Description

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()

Description

Returns the owner of the DTD implementation.

Syntax

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

getPublicId()

Description

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()

Description

Returns the root tag for the DTD.

Syntax

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

getSystemId()

Description

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()

Description

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

Syntax

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

normalize()

Description

Normalizes the DTD.

Syntax

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

printExternalDTD()

Description

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

Syntax	Description
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()

Description

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()

Description

Set the root tag for the DTD

Syntax

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

Parameter	Description
root	The root tag.

writeExternal()

Description

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.

ElementDecl Class

Description of ElementDecl

This class represents an element declaration in a DTD.

Syntax of ElementDecl

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

oracle.xml.parser.v2.ElementDecl

Implemented Interfaces of ElementDecl

```
java.io.Externalizable, java.io.Serializable
```

Fields of ElementDecl

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.

Table 2–5 Fields of ElementDecl (Cont.)

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

Methods of ElementDecl

Table 2–6 Summary of Methods of ElementDecl

Method	Description
ElementDecl() on page 2-26	Default constructor.
cloneNode() on page 2-26	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
expectedElements() on page 2-27	Returns vector of element names that can be appended to the element.
findAttrDecl() on page 2-27	Returns an attribute declaration object or null if not found
getAttrDecls() on page 2-27	Returns an enumeration of attribute declarations.
getContentElements() on page 2-28	Returns a vector of elements that can be appended to this element.
getContentType() on page 2-28	Returns the content model of element.
getNodeName() on page 2-28	Returns the name of the Element Decl.
getNodeType() on page 2-28	Returns a code representing the type of the underlying object.

Table 2–6 Summary of Methods of ElementDecl

Method	Description
getParseTree() on page 2-29	Returns the root node of Content Model Parse Tree.
readExternal() on page 2-29	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:
validateContent() on page 2-29	Validates the content of a element node.
writeExternal() on page 2-30	Saves the state of the object by creating a binary compressed stream with information about this object.

ElementDecl()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Returns an enumeration of attribute declarations.

Syntax

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

getContentElements()

Description

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

Syntax

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

getContentType()

Description

Returns the content model of element.

Syntax

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

getNodeName()

Description

Returs the name of the Element Decl.

Syntax

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

getNodeType()

Description

Returns a code representing the type of the underlying object.

Syntax

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

getParseTree()

Description

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()

Description

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()

Description

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()

Description

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

Description of XMLAttr

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

Syntax of XMLAttr

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

`oracle.xml.parser.v2.XMLAttr`

Implemented Interfaces of XMLAttr

`java.io.Externalizable, NSName, oracle.xml.util.NSName, java.io.Serializable`

Methods of XMLAttr

Table 2–7 Summary of Methods of XMLAttr

Method	Description
addText() on page 2-32	Adds text to the XMLNode.
cloneNode() on page 2-32	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
getExpandedName() on page 2-33	Returns the fully resolved Name for this attribute.
getLocalName() on page 2-33	Returns the local name of this attribute.
getName() on page 2-33	Returns the attribute name.
getNamespaceURI() on page 2-33	Returns the namespace of the attribute.
getNextAttribute() on page 2-34	Returns the next attribute, if any.
getNextSibling() on page 2-34	Returns the next sibling, if any.
getNodeType() on page 2-34	Returns a code representing the type of the underlying object.
getNodeValue() on page 2-34	Returns the value of this node, depending on its type.
getOwnerElement() on page 2-35	Returns the element which owns this attribute.
getParentNode() on page 2-35	Returns the parent of this node.

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

Method	Description
getPrefix() on page 2-35	Returns the name space prefix of the element.
getPreviousSibling() on page 2-35	Returns the previous sibling, if any.
getSpecified() on page 2-36	Returns TRUE if the attribute was specified explicitly in the element, FALSE otherwise.
getValue() on page 2-36	Returns the attribute value.
readExternal() on page 2-36	Restores the information written by writeExternal.
setNodeValue() on page 2-37	Sets the value of this node, depending on its type.
setValue() on page 2-37	Sets the value.
writeExternal() on page 2-37	Saves the state of the object in a binary compressed stream.

addText()

Description
Adds text to the XMLNode.

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

Parameter	Description
str	Text added.

cloneNode()

Description
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()

Description

Returns the fully resolved Name for this attribute.

Syntax

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

getLocalName()

Description

Returns the local name of this attribute.

Syntax

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

getName()

Description

Returns the attribute name.

Syntax

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

getNamespaceURI()

Description

Returns the namespace of the attribute.

Syntax

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

getNextAttribute()

Description

Returns the next attribute, if any.

Syntax

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

getNextSibling()

Description

Returns the next sibling, if any.

Syntax

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

getNodeType()

Description

Returns a code representing the type of the underlying object.

Syntax

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

getNodeValue()

Description

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()

Description

Returns the element which owns this attribute.

Syntax

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

getParentNode()

Description

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()

Description

Returns the name space prefix of the element.

Syntax

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

getPreviousSibling()

Description

Returns the previous sibling, if any.

Syntax

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

getSpecified()

Description

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

Syntax

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

getValue()

Description

Returns the attribute value.

Syntax

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

readExternal()

Description

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()

Description

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()

Description

Sets the value.

Syntax

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

Parameter	Description
val	The value to set.

writeExternal()

Description

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

Description of XMLCDATA

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

Syntax of XMLCDATA

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

`oracle.xml.parser.v2.XMLCDATA`

Implemented Interfaces of XMLCDATA

`java.io.Externalizable`, `java.io.Serializable`

Methods of XMLCDATA

Table 2–8 Summary of Methods of XMLCDATA

Method	Description
XMLCDATA() on page 2-39	Default constructor.
getNodeName() on page 2-40	Returns the name of the node.
getNodeType() on page 2-40	Returns a code representing the type of the underlying object.
readExternal() on page 2-40	Reads the information written in the compressed stream by `writeExternal()` method and restores the object correspondingly.
writeExternal() on page 2-41	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLCDATA()

Description

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()

Description

Returns the name of the node.

Syntax

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

getNodeType()

Description

Returns a code representing the type of the underlying object.

Syntax

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

readExternal()

Description

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()

Description

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

Description of XMLComment

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

Syntax of XMLComment

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

`oracle.xml.parser.v2.XMLComment`

Implemented Interfaces of XMLComment

`java.io.Externalizable, java.io.Serializable`

Methods of XMLComment

Table 2–9 Summary of XMLComment

Method	Description
XMLComment() on page 2-42	Default constructor. Note that this constructor is used only during deserialization / decompression of this DOM node.
addText() on page 2-43	Adds the comment text.
getNodeName() on page 2-43	Returns a name of the node.
getNodeType() on page 2-43	Returns a code representing the type of the underlying object.
readExternal() on page 2-44	Reads the information written in the compressed stream by writeExternal() method and restores the object correspondingly.
reportSAXEvents() on page 2-44	Reports SAX Events from a DOM Tree.
writeExternal() on page 2-44	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLComment()

Description

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()

Description

Adds the comment text.

Syntax

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

Parameter	Description
str	The comment text.

getNodeName()

Description

Returns a name of the node.

Syntax

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

getNodeType()

Description

Returns a code representing the type of the underlying object.

Syntax

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

readExternal()

Description

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()

Description

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

Syntax

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

Parameter	Description
cntHandler	Content handler.

writeExternal()

Description

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

Description of XMLDeclPI

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

Syntax of XMLDeclPI

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

oracle.xml.parser.v2.XMLPI
  |
  +--oracle.xml.parser.v2.XMLDeclPI
```

Implemented Interfaces of XMLDeclPI

`java.io.Externalizable, java.io.Serializable`

Methods of XMLDeclPI

Table 2–10 Summary of Methods of XMLDeclPI

Method	Description
XMLDeclPI() on page 2-47	Creates a new instance of XMLDeclPI.
cloneNode() on page 2-48	Returns a duplicate of this node; serves as a generic copy constructor.
getData() on page 2-48	Returns the fully constructed string 'version=1.0 ....'.
getEncoding() on page 2-48	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 2-49	Returns the value of this node.
getStandalone() on page 2-49	Returns the standalone information, or the standalone attribute stored in the <?xml ...?> tag.
getVersion() on page 2-49	Retrieves the version information, or the version number stored in the <?xml ...?> tag.
readExternal() on page 2-49	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly. Throws the following exceptions:

Table 2–10 Summary of Methods of XMLDeclPI (Cont.)

Method	Description
setEncoding() on page 2-50	Sets the character encoding for output.
setStandalone() on page 2-50	Sets the standalone information stored in the <?xml ...?> tag
setVersion() on page 2-51	Sets the version number stored in the <?xml ...?> tag.
writeExternal() on page 2-51	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLDeclPI()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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

Syntax

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

getVersion()

Description

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

Syntax

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

readExternal()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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

Description of XMLDocument

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 of XMLDocument

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

`oracle.xml.parser.v2.XMLDocument`

Implemented Interfaces of XMLDocument

```
java.io.Externalizable, java.io.Serializable
```

Methods of XMLDocument

Table 2–11 Summary of Methods of XMLDocument

Method	Description
XMLDocument() on page 2-55	Creates an empty document.
addID() on page 2-55	Adds an ID Element associated with this document.
adoptNode() on page 2-56	Adopts a node from another document to this document.
appendChild() on page 2-56	Appends a new node to the document. T
cloneNode() on page 2-57	Returns a duplicate of this node; serves as a generic copy constructor for nodes
createAttribute() on page 2-57	Creates an Attr of the given name.
createAttributeNS() on page 2-58	Creates an attribute with the given qualified name and namespace URI.
createCDATASection() on page 2-58	Creates a CDATASection node whose value is the specified string.

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

Method	Description
createComment() on page 2-59	Creates a Comment node given the specified string.
createDocumentFragment() on page 2-59	Creates an empty DocumentFragment object.
createElement() on page 2-59	Creates an element of the type specified.
createElementNS() on page 2-60	Creates an element of the given qualified name and namespace URI.
createEntityReference() on page 2-60	Creates an EntityReference object.
createEvent() on page 2-61	Creates an event object of the specified type.
createMutationEvent() on page 2-61	Creates a Mutation Event object of specified type.
createNodeIterator() on page 2-61	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 2-62	Creates a ProcessingInstruction node given the specified name and data strings. Throws DOMException:
createRange() on page 2-62	Creates a new Document Range Object, with Start and End Boundary points at the beginning of the document.
createRangeEvent() on page 2-63	Creates a Range Event object of specified type.
createTextNode() on page 2-63	Creates a Text node given the specified string.
createTraversalEvent() on page 2-63	Creates a Traversal Event object of specified type.
createTreeWalker() on page 2-64	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 2-64	Returns vector of element names that can be appended to the element.
getColumnNumber() on page 2-65	Returns column number debug information.
getDebugMode() on page 2-65	Returns the debug flag.
getDoctype() on page 2-65	Returns the Document Type Declaration (DTD) associated with this document.
getDocumentElement() on page 2-65	Accesses the child node that is the root element of the document.

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

Method	Description
getElementById() on page 2-66	Returns the Element whose ID is given by elementId.
getElementsByTagName() on page 2-66	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 2-66	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 2-67	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 2-67	Returns the ID element hashtable in the XML DOM Tree.
getImplementation() on page 2-67	Returns the DOMImplementation object that handles this document.
getLineNumber() on page 2-68	Returns line number debug information.
getNodeType() on page 2-68	Returns a code representing the type of the underlying object.
getOwnerDocument() on page 2-68	Returns the Document object associated with this node.
getStandalone() on page 2-68	Retrieves the standalone information; this is the standalone attribute stored in the <?xml ...?> tag.
getSystemId() on page 2-69	Returns the system id of the entity contain this node.
getText() on page 2-69	Returns the non-marked-up text contained by this element.
getVersion() on page 2-69	Retrieves the version information stored in the <?xml ...?> tag.
importNode() on page 2-69	Imports a node from another document to this document.
insertBefore() on page 2-70	Inserts the node newChild before the existing child node refChild.
print() on page 2-71	Writes the contents of this document to the given output.
printExternalDTD() on page 2-72	Writes the contents of this document to the given output stream.
readExternal() on page 2-72	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.

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

Method	Description
removeChild() on page 2-73	Removes the elem from this documents list of child nodes.
replaceChild() on page 2-73	Replaces the child node oldChild with newChild in the list of children, and returns the oldChild node.
reportSAXEvents() on page 2-74	Reports SAX Events from a DOM Tree.
setDoctype() on page 2-74	Sets the doctype URI for the document.
setEncoding() on page 2-75	Sets the character encoding for output.
setLocale() on page 2-75	Sets the locale for error reporting.
setNodeContext() on page 2-75	Sets node context.
setParsedDoctype() on page 2-76	Sets the doctype object by parsing sysid.
setStandalone() on page 2-76	Sets the standalone information stored in the <?xml ...?> tag.
setVersion() on page 2-77	Sets the version number stored in the <?xml ...?> tag.
validateElementContent() on page 2-77	Validates the content of a element node.
writeExternal() on page 2-77	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLDocument()

Description
Creates an empty document.

Syntax
```
public  XMLDocument();
```

addID()

Description
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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Creates an empty `DocumentFragment` object.

Syntax

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

createElement()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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.
data	The data for the node.

createRange()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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.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()

Description

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()

Description

Returns column number debug information.

Syntax

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

getDebugMode()

Description

Returns the debug flag.

Syntax

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

getDoctype()

Description

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()

Description

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

Syntax

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

getElementById()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Returns the ID element hashtable in the XML DOM Tree.

Syntax

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

getImplementation()

Description

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()

Description

Returns line number debug information.

Syntax

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

getNodeType()

Description

Returns a code representing the type of the underlying object.

Syntax

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

getOwnerDocument()

Description

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()

Description

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

Syntax

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

getSystemId()

Description

Returns the system id of the entity contain this node.

Syntax

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

getText()

Description

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()

Description

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

Syntax

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

importNode()

Description

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; this method creates a new copy of the source node.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). As in the cloneNode operation on a Node, the source node is not altered. 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.
deep	Indicates whether the descen.dants of this node are to be imported.

insertBefore()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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. This is an override of the `XMLNode.removeChild()` method. 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()

Description
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()

Description
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()

Description

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()

Description

Sets the locale for error reporting.

Syntax

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

Parameter	Description
locale	Locale for error reporting.

setNodeContext()

Description

Sets node context.

Syntax

```
public void setNodeContext( oracle.xml.util.NodeContext nctx);
```

Parameter	Description
nctx	The context to set.

setParsedDoctype()

Description

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()

Description

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

Syntax

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

Parameter	Description
value	The attribute value.

setVersion()

Description

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()

Description

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()

Description

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

Description of XMLDocumentFragment

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 of XMLDocumentFragment

```
public class XMLDocumentFragment implements java.io.Serializable
```

`oracle.xml.parser.v2.XMLDocumentFragment`

Implemented Interfaces of XMLDocumentFragment

```
java.io.Serializable
```

Methods of XMLDocumentFragment

Table 2–12 Summary of Methods of XMLDocumentFragment

Method	Description
getAttributes() on page 2-79	Returns the attributes of the XMLDocumentFragment.
getNodeType() on page 2-80	Returns a code representing the type of the underlying object.
getParentNode() on page 2-80	Returns the parent of this node.

getAttributes()

Description

Returns the attributes of the XMLDocumentFragment - an empty NamedNodeMap.

Syntax

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

getNodeType()

Description
Returns a code representing the type of the underlying object.

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

getParentNode()

Description
Returns the parent of this node.

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

XMLDOMException Class

Description of XMLDOMException

This class is used to throw DOM exceptions.

Syntax of XMLDOMException

```
public class XMLDOMException
```

`oracle.xml.parser.v2.XMLDOMException`

Methods of XMLDOMException

XMLDOMException()

Description

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

Description of XMLDOMImplementation

This class implements the DOMImplementation

Syntax of XMLDOMImplementation

```
public class XMLDOMImplementation implements java.io.Serializable
```

`oracle.xml.parser.v2.XMLDOMImplementation`

Implemented Interfaces of XMLDOMImplementation

`java.io.Serializable`

Methods of XMLDOMImplementation

Table 2–13 Summary of Methods of XMLDOMImplementation

Method	Description
XMLDOMImplementation() on page 2-82	Creates a new instance of XMLDOMImplementation.
createDocument() on page 2-83	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 2-83	Creates an empty DocumentType node with root element name and system / public identifier.
hasFeature() on page 2-84	Tests if the DOM implementation implements a specific feature.
setFeature() on page 2-84	Sets a specified feature.

XMLDOMImplementation()

Description

Creates a new instance of XMLDOMImplementation.

Syntax

```
public  XMLDOMImplementation();
```

createDocument()

Description

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()

Description

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()

Description

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()

Description

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

Description of XMLElement

This class implements the DOM `Element` Interface.

Syntax of XMLElement

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

`oracle.xml.parser.v2.XMLElement`

Implemented Interfaces of XMLElement

`java.io.Externalizable, NSName, oracle.xml.util.NSName, NSResolver, java.io.Serializable`

Methods of XMLElement

Table 2–14 Summary of Methods of XMLElement

Method	Description
XMLElement() on page 2-88	Default constructor.
cloneNode() on page 2-88	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
getAttribute() on page 2-89	Returns an attribute value by name; if that attribute does not have a specified or default value, returns an empty string.
getAttributeNode() on page 2-89	Returns an Attr node by name, or NULL if there is no such attribute.
getAttributeNodeNS() on page 2-89	Returns attribute with the given namespaceURI and localName if it exists; otherwise, returns NULL.
getAttributeNS() on page 2-90	Returns the value of the attribute with namespace URI and localName, if it exists; threshed, returns NULL.
getAttributes() on page 2-90	Returns a NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.
getChildrenByTagName() on page 2-91	Returns a NodeList of all immediate children with a given tag name. The options are described in the following table.

Table 2–14 Summary of Methods of XMLElement (Cont.)

Method	Description
getElementsByTagName() on page 2-91	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 2-92	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 2-92	Returns the fully resolved name for this element.
getFirstAttribute() on page 2-89	Retrieves the first Attr node, or NULL if there is no attribute.
getLocalName() on page 2-92	Returns the local Name for this element.
getNamespaceURI() on page 2-93	Returns the name space URI of this element.
getNodeType() on page 2-93	Returns a code representing the type of the underlying object.
getPrefix() on page 2-93	Returns the namespace prefix for this element.
getQualifiedName() on page 2-93	Returns the qualified name for this element.
getTagName() on page 2-94	Returns the name of the element.
hasAttribute() on page 2-94	Returns TRUE when an attribute with a given name is specified on this element or has a default value.
hasAttributeNS() on page 2-94	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 2-95	Returns TRUE if this node has any attributes.
readExternal() on page 2-95	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 2-95	Removes an attribute by name.
removeAttributeNode() on page 2-96	Removes and returns the specified attribute.
removeAttributeNS() on page 2-96	Removes an attribute by local name and namespace URI.
reportSAXEvents() on page 2-97	Reports SAX Events from a DOM Tree.
resolveNamespacePrefix() on page 2-97	Finds the namespace definition in scope in this element, given a namespace prefix.
setAttribute() on page 2-97	Adds a new attribute.

Table 2–14 Summary of Methods of XMLElement (Cont.)

Method	Description
setAttributeNode() on page 2-98	Adds a new attribute node.
setAttributeNodeNS() on page 2-99	Adds a new namespace aware attribute node.
validateContent() on page 2-100	Validates the content of a element node.
writeExternal() on page 2-101	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLElement()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Returns a `NodeList` of all the `Element`s 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()

Description

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()

Description

Returns the fully resolved name for this element.

Syntax

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

getFirstAttribute()

Description

Retrieves the first `Attr` node, or `NULL` if there is no attribute.

Syntax

```
public XMLNode getFirstAttribute();
```

getLocalName()

Description

Returns the local Name for this element.

Syntax

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

getNamespaceURI()

Description

Returns the name space URI of this element.

Syntax

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

getNodeType()

Description

Returns a code representing the type of the underlying object.

Syntax

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

getPrefix()

Description

Returns the namespace prefix for this element.

Syntax

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

getQualifiedName()

Description

Returns the qualified name for this element.

Syntax

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

getTagName()

Description

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()

Description

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()

Description

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()

Description

Returns `TRUE` if this node has any attributes, `FALSE` otherwise.

Syntax

```
public boolean hasAttributes();
```

readExternal()

Description

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()

Description

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()

Description

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()

Description

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.

Parameter	Description
localName	Local name of the attribute to be removed.

reportSAXEvents()

Description

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()

Description

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()

Description

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 raised 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()

Description

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()

Description

Adds a new attribute. 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 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.

Syntax	Description
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()

Description

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.
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()

Description

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

Description of XMLEntity

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 of XMLEntity

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

`oracle.xml.parser.v2.XMLEntity`

Implemented Interfaces of XMLEntity

`java.io.Externalizable, java.io.Serializable`

Methods of XMLEntity

Table 2–15 Summary of Methods of XMLEntity

Method	Description
XMLEntity() on page 2-103	Default constructor.
cloneNode() on page 2-103	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
getNodeType() on page 2-103	Returns a code representing the type of the underlying object.
getNodeValue() on page 2-104	Returns the value of this node, depending on its type.
getNotationName() on page 2-104	Returns the name of the notation for an unparsed the entity. For parsed entities, this is `null`.
getPublicId() on page 2-104	Returns the public identifier.
getSystemId() on page 2-104	Returns the system identifier.
readExternal() on page 2-105	Reads the information written in the compressed stream by writeExternal() method and restores the object correspondingly.
setNodeValue() on page 2-105	Sets the value of entity.
writeExternal() on page 2-105	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLEntity()

Description

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()

Description

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()

Description

Returns a code representing the type of the underlying object.

Syntax

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

getNodeValue()

Description

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()

Description

Returns the name of the notation for an unparsed the entity. For parsed entities, this is `null`.

Syntax

```
public String getNotationName();
```

getPublicId()

Description

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()

Description

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()

Description

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()

Description

Sets the value of entity.

Syntax

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

Parameter	Description
arg	The new value of the entity.

writeExternal()

Description

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

Description of XMLEntityReference

This class implements DOM `EntityReference` interface.

Syntax of XMLEntityReference

```
public class XMLEntityReference implements java.lang.Cloneable,
java.io.Externalizable
```

`oracle.xml.parser.v2.XMLEntityReference`

Implemented Interfaces of XMLEntityReference

`java.lang.Cloneable, java.io.Externalizable, java.io.Serializable`

Methods of XMLEntityReference

Table 2–16 Summary of Methods of XMLEntityReference

Method	Description
XMLEntityReference() on page 2-107	Default constructor.
getNodeType() on page 2-108	Returns a code representing the type of the underlying object.
readExternal() on page 2-108	Reads the information written in the compressed stream by writeExternal() method and restores the object correspondingly.
writeExternal() on page 2-108	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLEntityReference()

Description

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()

Description

Returns a code representing the type of the underlying object.

Syntax

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

readExternal()

Description

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()

Description

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

Description of XMLNode

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 of XMLNode

```
public abstract class XMLNode implements java.lang.Cloneable,
java.io.Externalizable
```

oracle.xml.parser.v2.XMLNode

Subclasses of XMLNode

XMLNSNode

Implemented Interfaces of XMLNode

`java.lang.Cloneable, java.io.Externalizable, java.io.Serializable`

Fields of XMLNode

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.

Table 2–17 Fields of XMLNode (Cont.)

Field	Syntax	Description
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	CharacterData within a node has been modified.
DOMNodeInserted	public static final String DOMNodeInserted	Node has been added as a child of another node.
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.

Table 2–17 Fields of XMLNode (Cont.)

Field	Syntax	Description
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
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

Methods of XMLNode

Table 2–18 Summary of Methods of XMLNode

Method	Description
XMLNode() on page 2-115	Constructs a new XMLNode.
addEventListener() on page 2-115	Registers event listeners on the event target (node).
appendChild() on page 2-116	Adds the node newChild to the end of the list of children of this node, and returns the new node.
cloneNode() on page 2-116	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
dispatchEvent() on page 2-117	Dispatches events into the implementations event model.

Table 2–18 Summary of Methods of XMLNode (Cont.)

Method	Description
getAttributes() on page 2-117	Returns a NamedNodeMap containing the attributes of this node.
getChildNodes() on page 2-117	Returns all children of this node in a NodeList.
getColumnNumber() on page 2-118	Returns the column number debug information.
getDebugMode() on page 2-118	Returns the debug information mode.
getFirstChild() on page 2-118	Returns the first child of this node.
getLastChild() on page 2-118	Returns the last child of this node.
getLineNumber() on page 2-119	Returns the line number debug information.
getLocalName() on page 2-119	Returns the Local Name of this node overrided by node types for which namespace is meaningful.
getNamespaceURI() on page 2-119	Returns the namespace URI of this node, overrided by node types for which namespace is meaningful.
getNextSibling() on page 2-119	Returns the node immediately following this node.
getNodeName() on page 2-120	Returns the name of the node.
getNodeType() on page 2-120	Returns the type of the node.
getNodeValue() on page 2-120	Returns the value of this node, depending on its type.
getOwnerDocument() on page 2-120	Returns the Document object associated with this node.
getParentNode() on page 2-121	Returns the parent of this node.
getPrefix() on page 2-121	Returns the prefix of this node overrided by node types for which namespace is meaningful.
getPreviousSibling() on page 2-121	Returns the node immediately preceding this node. I
getProperty() on page 2-121	Returns the value of a property of the node.
getSystemId() on page 2-122	Returns the system id of the entity containing this node.
getText() on page 2-122	Returns the non-marked-up text contained by this element.
hasAttributes() on page 2-122	Determines whether this node (if it is an element) has any attributes.
hasChildNodes() on page 2-122	Determines whether a node has any children.

Table 2–18 Summary of Methods of XMLNode (Cont.)

Method	Description
insertBefore() on page 2-123	Inserts the node newChild before the existing child node refChild.
isNodeFlag() on page 2-123	Returns TRUE if the node flag information is set.
isSupported() on page 2-124	Tests whether the DOM implementation implements a specific feature and that feature is supported by this node.
print() on page 2-124	Writes the contents of this node into output.
readExternal() on page 2-125	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
removeChild() on page 2-125	Removes the child node indicated by oldChild from the list of children.
removeEventListener() on page 2-126	Removes event listeners from the event target (node).
replaceChild() on page 2-126	Replaces the child node oldChild with newChild in the list of children, and returns the replaced node.
reportSAXEvents() on page 2-127	Reports SAX Events from a DOM Tree. Throws SAXException.
resetNodeFlag() on page 2-127	Resets the node flag information.
selectNodes() on page 2-127	Returns nodes from the tree which match the given pattern, as a NodeList.
selectSingleNode() on page 2-128	Returns the first node from the tree that matches the given pattern.
setDebugInfo() on page 2-129	Sets debug information in the node.
setNodeFlag() on page 2-129	Sets the node flag information.
setNodeValue() on page 2-130	Sets the value of this node, depending on its type.
setPrefix() on page 2-130	Sets the prefix of this node overrided by node types for which namespace is meaningful.
setProperty() on page 2-130	Sets a property of the node.
transformNode() on page 2-131	Transforms a node in the tree using the given stylesheet, and returns the resulting DocumentFragment.
valueOf() on page 2-131	Selects the value of the first node from tree that matches the pattern.

Table 2–18 Summary of Methods of XMLNode (Cont.)

Method	Description
writeExternal() on page 2-132	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLNode()

Description

Constructs a new `XMLNode`.

Syntax

```
protected  XMLNode();
```

Parameter	Description
tag	Name of the node.

addEventListener()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Returns the column number debug information.

Syntax

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

getDebugMode()

Description

Returns the debug information mode.

Syntax

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

getFirstChild()

Description

Returns the first child of this node. If there is no such node, this returns `null`.

Syntax

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

getLastChild()

Description

Returns the last child of this node. If there is no such node, this returns `null`.

Syntax

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

getLineNumber()

Description

Returns the line number debug information.

Syntax

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

getLocalName()

Description

Returns the Local Name of this node overrided by node types for which namespace is meaningful.

Syntax

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

getNamespaceURI()

Description

Returns the namespace URI of this node, overrided by node types for which namespace is meaningful.

Syntax

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

getNextSibling()

Description

Returns the node immediately following this node. If there is no such node, this returns `null`.

Syntax

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

getNodeName()

Description
Returns the name of the node.

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

getNodeType()

Description
Returns the type of the node.

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

getNodeValue()

Description
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()

Description
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()

Description

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()

Description

Returns the prefix of this node overrided by node types for which namespace is meaningful.

Syntax

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

getPreviousSibling()

Description

Returns the node immediately preceding this node. If there is no such node, this returns `null`.

Syntax

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

getProperty()

Description

Returns the value of a property of the node.

Syntax

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

Parameter	Description
propName	Name of the property.

getSystemId()

Description

Returns the system id of the entity containing this node.

Syntax

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

getText()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Returns `TRUE` if the node flag information is set.

Syntax

```
public boolean isNodeFlag( int flag);
```

Parameter	Description
flag	The flag.

isSupported()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Resets the node flag information.

Syntax

```
public void resetNodeFlag( int flag);
```

Parameter	Description
flag	The node flag.

selectNodes()

Description

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()

Description

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.

Parameter	Description
nsr	NSResolver to resolve any prefixes that occur in given pattern.

setDebugInfo()

Description

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()

Description

Sets the node flag information.

Syntax

```
public void setNodeFlag( int flag);
```

Parameter	Description
flag	The node flag.

setNodeValue()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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.
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()

Description

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

Description of XMLNotation

This class implements the DOM `Notation` interface and represents a notation declared in the Document Type Definition.

Syntax of XMLNotation

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

`oracle.xml.parser.v2.XMLNotation`

Implemented Interfaces of XMLNotation

`java.io.Externalizable, java.io.Serializable`

Methods of XMLNotation

Table 2–19 Summary of Methods of XMLNotation

Method	Description
XMLNotation() on page 2-134	Default constructor.
cloneNode() on page 2-134	Returns a duplicate of this node; serves as a generic copy constructor for nodes.
getNodeName() on page 2-134	Returns the name of the Notation.
getNodeType() on page 2-135	Returns a code representing the type of the underlying object.
getPublicId() on page 2-135	Returns the Public identifier; if not specified, then null.
getSystemId() on page 2-135	Returns the System identifier; if not specified, then null.
readExternal() on page 2-135	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
setPublicId() on page 2-136	Sets the Public Identifier.
setSystemId() on page 2-136	Sets the System Identifier.
writeExternal() on page 2-136	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLNotation()

Description

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()

Description

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`, clone only the node itself (and its attributes, if it is an `Element`)

getNodeName()

Description

Returns the name of the Notation

Syntax

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

getNodeType()

Description

Returns a code representing the type of the underlying object.

Syntax

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

getPublicId()

Description

Returns the Public identifier; if not specified, then `null`.

Syntax

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

getSystemId()

Description

Returns the System identifier; if not specified, then `null`.

Syntax

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

readExternal()

Description

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()

Description

Sets the Public Identifier.

Syntax

```
public void setPublicId( String pubid);
```

Parameter	Description
pubid	Public Identifier to set.

setSystemId()

Description

Sets the System Identifier.

Syntax

```
public void setSystemId( String url);
```

Parameter	Description
url	System identifier to set.

writeExternal()

Description

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

Description of XMLNSNode

Extends XMLNode to add support for Namespace names and children

Syntax of XMLNSNode

```
public class XMLNSNode extends oracle.xml.parser.v2.XMLNode

oracle.xml.parser.v2.XMLNode
  |
  +--oracle.xml.parser.v2.XMLNSNode
```

Implemented Interfaces of XMLNSNode

`java.lang.Cloneable, java.io.Externalizable, java.io.Serializable`

Methods of XMLNSNode

Table 2–20 Summary of Methods XMLNSNode

Method	Description
XMLNSNode() on page 2-139	Constructs a new XMLNSNode.
addText() on page 2-139	Adds text to this node, or appends it to the last child if the last child is a text node.
appendChild() on page 2-140	Adds the node newChild to the end of the list of children of this node.
getChildNodes() on page 2-141	Returns all children of this node as a NodeList.
getFirstChild() on page 2-141	Returns the first child of this node.
getLastChild() on page 2-141	Returns the last child of this node.
getLocalName() on page 2-142	Returns the Local Name of this node overrided by node types for which namespace is meaningful.
getNamespaceURI() on page 2-142	Returns the namespace URI of this node overrided by node types for which namespace is meaningful.
getNodeName() on page 2-142	Returns the name of this node, depending on its type.
getPrefix() on page 2-142	Returns the prefix of this node overrided by node types for which namespace is meaningful.

Table 2–20 Summary of Methods XMLNSNode (Cont.)

Method	Description
getText() on page 2-143	Returns the non-marked-up text contained by this element.
hasChildNodes() on page 2-143	Determines whether a node has any children.
insertBefore() on page 2-143	Inserts the child node before an existing child node.
normalize() on page 2-144	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.
removeChild() on page 2-144	Removes the child node indicated by oldChild from the list of children.
replaceChild() on page 2-144	Replaces the child node oldChild with newChild in the list of children.
setPrefix() on page 2-145	Sets the prefix of this node overrided by node types for which namespace is meaningful.

XMLNSNode()

Description

Constructs a new `XMLNSNode`.

Syntax

```
protected  XMLNSNode( String tag);
```

Parameter	Description
tag	Name of the node.

addText()

Description

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()

Description

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()

Description

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()

Description

Returns the first child of this node. If there is no such node, this returns `null`.

Syntax

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

getLastChild()

Description

Returns the last child of this node. If there is no such node, this returns `null`.

Syntax

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

getLocalName()

Description

Returns the Local Name of this node overrided by node types for which namespace is meaningful.

Syntax

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

getNamespaceURI()

Description

Returns the namespace URI of this node. overrided by node types for which namespace is meaningful.

Syntax

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

getNodeName()

Description

Returns the name of this node, depending on its type

Syntax

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

getPrefix()

Description

Returns the prefix of this node overrided by node types for which namespace is meaningful.

Syntax

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

getText()

Description

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()

Description

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()

Description

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.
refChild	The reference node, or the node before which the new node must be inserted.

normalize()

Description

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()

Description

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()

Description

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 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.

setPrefix()

Description

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

Description of XMLOutputStream

This class writes output stream and can handle XML encoding.

Syntax of XMLOutputStream

```
public class XMLOutputStream extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.parser.v2.XMLOutputStream
```

Fields of XMLOutputStream

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 readibility.

Methods of XMLOutputStream

Table 2–22 Summary of Methods of XMLOutputStream

Method	Description
XMLOutputStream() on page 2-148	Builds an ASCII output.
addIndent() on page 2-148	Sets indenting level for output.
close() on page 2-149	Closes the output stream.
flush() on page 2-149	Flushes the output stream.
getOutputStyle() on page 2-149	Returns the current output style.
setEncoding() on page 2-149	Sets the output character encoding.
setOutputStyle() on page 2-150	Sets the Output the style.
write() on page 2-150	Outputs character according to type of the output stream.

Table 2–22 Summary of Methods of XMLOutputStream (Cont.)

Method	Description
writeChars() on page 2-151	Writes string to the output.
writeIndent() on page 2-151	Writes an indentation.
writeNewLine() on page 2-151	Writes a new line.
writeQuotedString() on page 2-151	Writes string with surrounding quotes.

XMLOutputStream()

Description

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()

Description

Sets indenting level for output.

Syntax

```
public void addIndent( int offset);
```

Parameter	Description
offset	The indenting level.

close()

Description

Closes the output stream. Throws `IOException` if there is any error.

Syntax

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

flush()

Description

Flushes the output stream. Throws `IOException` if there is any error.

Syntax

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

getOutputStyle()

Description

Returns the current output style.

Syntax

```
public int getOutputStyle();
```

setEncoding()

Description

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()

Description

Sets the Output the style.

Syntax

```
public void setOutputStyle( int style);
```

Parameter	Description
s	The output style

write()

Description

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()

Description

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()

Description

Writes an indentation. Throws `IOException` if there is any error in writing the string.

Syntax

```
public void writeIndent();
```

writeNewLine()

Description

Writes a new line. Throws `IOException` if there is any error in writing the string.

Syntax

```
public void writeNewLine();
```

writeQuotedString()

Description

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

Description of XMLAPI

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

Syntax of XMLAPI

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

`oracle.xml.parser.v2.XMLPI`

Subclasses of XMLAPI

`XMLDeclPI`

Implemented Interfaces of XMLAPI

`java.io.Externalizable, java.io.Serializable`

Methods XMLPI

Table 2–23 Summary of Methods of XMLPI

Method	Description
XMLPI() on page 2-154	Creates a new instance of XMLPI.
addText() on page 2-154	Adds text string to the node, and returns the updated node.
getNodeName() on page 2-154	Returns the name of the PI Node.
getNodeType() on page 2-154	Returns the type of node of the underlying object.
getTarget() on page 2-155	Returns the target of this PI.
readExternal() on page 2-155	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
reportSAXEvents() on page 2-155	Reports SAX Events from a DOM Tree.
writeExternal() on page 2-156	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLPI()

Description

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()

Description

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()

Description

Returns the name of the PI Node.

Syntax

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

getNodeType()

Description

Returns the type of node of the underlying object

Syntax

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

getTarget()

Description

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()

Description

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()

Description

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

Syntax

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

Parameter	Description
cntHandler	Content handler.

writeExternal()

Description

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

Description of XMLPrintDriver

The `XMLPrintDriver` implements `PrintDriver` interface.

Syntax of XMLPrintDriver

```
public class XMLPrintDriver extends Object implements
oracle.xml.parser.v2.PrintDriver

java.lang.Object
  |
  +--oracle.xml.parser.v2.XMLPrintDriver
```

Implemented Interfaces of XMLPrintDriver

`PrintDriver`

Fields of XMLPrintDriver of XMLPrintDriver

Table 2–24 Fields of XMLPrintDriver

Field	Syntax	Description
`out`	`protected XMLOutputStream out`	XMLOutputStream object

Methods of XMLPrintDriver

Table 2–25 Summary of Methods of XMLPrintDriver

Method	Description
XMLPrintDriver() on page 2-158	Creates an instance of XMLPrintDriver.
close() on page 2-159	Closes the output stream or print writer
flush() on page 2-159	Flushes the output stream or print writer.
printAttribute() on page 2-159	Prints an XMLAttr node.
printAttributeNodes() on page 2-159	Calls print method for each attribute of the XMLElement.
printCDATASection() on page 2-160	Prints an XMLCDATA node.

Table 2–25 Summary of Methods of XMLPrintDriver (Cont.)

Method	Description
printChildNodes() on page 2-160	Calls print method for each child of the XMLNode.
printComment() on page 2-160	Prints an XMLComment node.
printDoctype() on page 2-161	Prints a DTD.
printDocument() on page 2-161	Prints an XMLDocument.
printDocumentFragment() on page 2-161	Prints an empty XMLDocumentFragment object.
printElement() on page 2-162	Prints an XMLElement.
printEntityReference() on page 2-162	Prints an XMLEntityReference node
printProcessingInstruction() on page 2-162	Prints an XMLPI node.
printTextNode() on page 2-163	Prints an XMLText node.
setEncoding() on page 2-163	Sets the encoding of the print driver.

XMLPrintDriver()

Description

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()

Description

Closes the output stream or print writer

Syntax

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

flush()

Description

Flushes the output stream or print writer.

Syntax

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

printAttribute()

Description

Prints an `XMLAttr` node.

Syntax

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

Parameter	Description
attr	The XMLAttr Node.

printAttributeNodes()

Description

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()

Description

Prints an `XMLCDATA` node.

Syntax

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

Parameter	Description
cdata	The XMLCDATA node

printChildNodes()

Description

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()

Description

Prints an `XMLComment` node.

Syntax

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

Parameter	Description
comment	The comment node.

printDoctype()

Description

Prints an DTD.

Syntax

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

Parameter	Description
dtd	The DTD to be printed.

printDocument()

Description

Prints an XMLDocument.

Syntax

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

Parameter	Description
doc	The document to be printed

printDocumentFragment()

Description

Prints an empty XMLDocumentFragment object.

Syntax

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

Parameter	Description
dfrag	The document fragment to be printed.

printElement()

Description

Prints an `XMLElement`.

Syntax

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

Parameter	Description
elem	The element to be printed.

printEntityReference()

Description

Prints an `XMLEntityReference` node

Syntax

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

Parameter	Description
en	The XMLEntityReference node.

printProcessingInstruction()

Description

Prints an `XMLPI` node

Syntax

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

Parameter	Description
pi	The XMLPI node.

printTextNode()

Description

Prints an `XMLText` node.

Syntax

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

Parameter	Description
text	The text node.

setEncoding()

Description

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

Description of XMLRangeException

This class customizes the RangeException.

Syntax of XMLRangeException

```
public class XMLRangeException
```

`oracle.xml.parser.v2.XMLRangeException`

Methods of XMLRangeException

XMLRangeException()

Description

Generates an XMLRangeException instance.

Syntax

```
public  XMLRangeException(short code);
```

Parameter	Description
code	

XMLText Class

Description of XMLText

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

Syntax of XMLText

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

`oracle.xml.parser.v2.XMLText`

Implemented Interfaces of XMLText

`java.io.Externalizable, java.io.Serializable`

Methods of XMLText

Table 2–26 Summary of Methods of XMLText

Method	Description
XMLText() on page 2-166	Creates an instance of XMLText.
addText() on page 2-166	Adds text to the data of the text node.
getData() on page 2-167	Returns the character data of the node that implements this interface.
getNodeName() on page 2-167	Returns the name of the XMLText Node.
getNodeType() on page 2-167	Returns a type of node representing the type of the underlying object.
getNodeValue() on page 2-168	Returns String value of this text node.
isWhiteSpaceNode() on page 2-168	Checks if the text node is a whitespace node.
readExternal() on page 2-168	Reads the information written in the compressed stream by writeExternal method and restores the object correspondingly.
reportSAXEvents() on page 2-169	Reports SAX Events from a DOM Tree.
splitText() on page 2-169	Breaks Text node into two Text nodes at specified offset.

Table 2–26 Summary of Methods of XMLText (Cont.)

Method	Description
writeExternal() on page 2-170	Saves the state of the object by creating a binary compressed stream with information about this object.

XMLText()

Description

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()

Description

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()

Description

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()

Description

Returns the name of the XMLText Node.

Syntax

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

getNodeType()

Description

Returns a type of node representing the type of the underlying object.

Syntax

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

getNodeValue()

Description

Returns String value of this text node. Throws `DOMException` if any error occurs when retrieving the value.

Syntax

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

isWhiteSpaceNode()

Description

Checks if the text node is a whitespace node.

Syntax

```
public boolean isWhiteSpaceNode();
```

readExternal()

Description

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()

Description

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

Syntax

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

Parameter	Description
cntHandler	Content handler.

splitText()

Description

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()

Description

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.

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:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

JXDocumentBuilder Class

Description of JXDocumentBuilder

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 of JXDocumentBuilder

```
public class JXDocumentBuilder
```

`oracle.xml.jaxp.JXDocumentBuilder`

Methods of 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 3-4	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 3-5	Specifies the `ErrorHandler` to be used to resolve entities present in the XML document to be parsed.

getDOMImplementation()

Description

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()

Description

Indicates whether or not this parser is configured to understand namespaces.

Syntax

```
public boolean isNamespaceAware();
```

isValidating()

Description

Indicates whether or not this parser is configured to validate XML documents.

Syntax

```
public boolean isValidating();
```

newDocument()

Description

Obtains a new instance of a DOM Document object with which to build a DOM tree.

Syntax

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

parse()

Description

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()

Description

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()

Description

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

Description of JXDocumentBuilderFactory

Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.

Syntax of JXDocumentBuilderFactory

```
public class JXDocumentBuilderFactory
```

`oracle.xml.jaxp.JXDocumentBuilderFactory`

Fields of 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	Sets Debug Mode - Boolean.TRUE or Boolean.FALSE.
DTD_OBJECT	public static final java.lang.String DTD_OBJECT	DTD Object to be used for validation.
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.
SCHEMA_OBJECT	public static final java.lang.String SCHEMA_OBJECT	Schema Object to be used for validation.
SHOW_WARNINGS	public static final java.lang.String SHOW_WARNINGS	Boolean to ignore warnings - Boolean.TRUE or Boolean.FALSE.

Table 3–2 Fields of JXDocumentBuilderFactory (Cont.)

Field	Syntax	Description
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.

Methods of JXDocumentBuilderFactory

Table 3–3 Summary of Methods of JXDocumentBuilderFactory

Method	Description
JXDocumentBuilderFactory() on page 3-7	Default constructor.
getAttribute() on page 3-8	Allows the user to retrieve specific attributes on the underlying implementation
isExpandEntityReferences() on page 3-8	Indicates if the factory is configured to produce parsers which expand entity reference nodes.
isIgnoringComments() on page 3-8	Indicates if the factory is configured to produce parsers which ignore comments.
isNamespaceAware() on page 3-9	Indicates if the factory is configured to produce namespace aware parsers.
newDocumentBuilder() on page 3-9	Creates a new instance of a DocumentBuilder using the currently configured parameters.
setAttribute() on page 3-9	Sets specific attributes on the underlying implementation.

JXDocumentBuilderFactory()

Description

Default constructor.

Syntax

```
public  JXDocumentBuilderFactory();
```

getAttribute()

Description

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 Object getAttribute( String name);
```

Parameter	Description
name	The name of the attribute.

isExpandEntityReferences()

Description

Indicates whether or not the factory is configured to produce parsers which expand entity reference nodes. Always returns `TRUE; c`urrently, there is no way to prevent entity references expansions.

Syntax

```
public boolean isExpandEntityReferences();
```

isIgnoringComments()

Description

Indicates whether or not the factory is configured to produce parsers which ignore comments. Always returns `FALSE;` currently, ignoring comments is not configurable.

Syntax

```
public boolean isIgnoringComments();
```

isNamespaceAware()

Description

Indicates whether or not the factory is configured to produce parsers which are namespace aware. Always returns `TRUE; currently`, there is no way to turn of Namespaces.

Syntax

```
public boolean isNamespaceAware();
```

newDocumentBuilder()

Description

Creates a new instance of a DocumentBuilder using the currently configured parameters. Throws `ParserConfigurationException` if a DocumentBuilder which satisfies the configuration requested cannot be created.

Syntax

```
public DocumentBuilder newDocumentBuilder();
```

setAttribute()

Description

Sets specific attributes on the underlying implementation. Throws `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.

JXSAXParser Class

Description of JXSAXParser

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 of JXSAXParser

```
public class JXSAXParser
```

`oracle.xml.jaxp.JXSAXParser`

Methods of JXSAXParser

Table 3–4 Summary of Methods of JXSAXParser

Method	**Description**
getProperty() on page 3-11	Returns the value of the requested property for in the underlying implementation of XMLReader.
getXMLReader() on page 3-11	Returns the XMLReader that is encapsulated by the implementation of this class.
isNamespaceAware() on page 3-12	Indicates if this parser is configured to understand namespaces.

Table 3–4 Summary of Methods of JXSAXParser (Cont.)

Method	Description
isValidating() on page 3-12	Indicates if this parser is configured to validate XML documents.
setProperty() on page 3-12	Sets the particular property in the underlying implementation of XMLReader

getProperty()

Description

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()

Description

Returns the XMLReader that is encapsulated by the implementation of this class.

Syntax

```
public XMLReader getXMLReader();
```

isNamespaceAware()

Description

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()

Description

Indicates whether or not this parser is configured to validate XML documents.

Syntax

```
public boolean isValidating();
```

setProperty()

Description

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

Description of JXSAXParserFactory

Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents.

Syntax

```
public class JXSAXParserFactory
```

oracle.xml.jaxp.JXSAXParserFactory

Methods of JXSAXParserFactory

Table 3–5 Summary of Methods of JXSAXParserFactory

Method	Description
JXSAXParserFactory() on page 3-13	Default constructor.
getFeature() on page 3-14	Returns the value of the requested property for in the underlying implementation of XMLReader.
isNamespaceAware() on page 3-14	Indicates if the factory is configured to produce namespace aware parsers.
newSAXParser() on page 3-14	Creates a new instance of a SAXParser using the currently configured factory parameters.
setFeature() on page 3-15	Sets the particular feature in the underlying implementation of XMLReader.

JXSAXParserFactory()

Description

Default constructor.

Syntax

```
public  JXSAXParserFactory();
```

getFeature()

Description

Returns the value of the requested property for in the underlying implementation of 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 boolean getFeature( String name);
```

Parameter	Description
name	The name of the property to be retrieved.

isNamespaceAware()

Description

Indicates whether or not the factory is configured to produce namespace aware parsers. Returns `TRUE` if the factory is configured for namespace aware parsers, `FALSE` otherwise.

Syntax

```
public boolean isNamespaceAware();
```

newSAXParser()

Description

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()

Description

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

Description of JXSAXTransformerFactory

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 abstract class (in our case, it is 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 of JXSAXTransformerFactory

```
public class JXSAXTransformerFactory
```

`oracle.xml.jaxp.JXSAXTransformerFactory`

Methods of JXSAXTransformerFactory

Table 3–6 Summary of Methods of JXSAXTransformerFactory

Method	Description
JXSAXTransformerFactory() on page 3-17	The default constructor.
getAssociatedStylesheet() on page 3-17	Retrieves the stylesheet specification(s) associated through the xml-stylesheet processing instruction.
getAttribute() on page 3-18	Allows the user to retrieve specific attributes on the underlying implementation.
getErrorListener() on page 3-18	Retrieves the current error event handler (which should never be NULL) for the TransformerFactory.
getFeature() on page 3-19	Looks up the value of a feature.

Table 3–6 Summary of Methods of JXSAXTransformerFactory (Cont.)

Method	Description
getURIResolver() on page 3-19	Retrieves the object used by default during the transformation to resolve URIs.
newTemplates() on page 3-19	Processes the Source into a Templates object, which is a compiled representation of the source.
newTemplatesHandler() on page 3-20	Retrieves a TemplatesHandler object that can process SAX ContentHandler events into a Templates object.
newTransformer() on page 3-20	Generates a new Transformer object.
newTransformerHandler() on page 3-21	Generates a TransformerHandler object.
newXMLFilter() on page 3-21	Creates an XMLFilter.
setAttribute() on page 3-22	Sets specific attributes on the underlying implementation.
setErrorListener() on page 3-23	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 3-23	Sets an object to use by default during the transformation to resolve URIs used in xsl:import, or xsl:include.

JXSAXTransformerFactory()

Description

The default constructor.

Syntax

```
public  JXSAXTransformerFactory();
```

getAssociatedStylesheet()

Description

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()

Description

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()

Description

Retrieves the current error event handler (which should never be `NULL`) for the TransformerFactory.

Syntax

```
public ErrorListener getErrorListener();
```

getFeature()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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);	Generates a TransformerHandler object that can process SAX ContentHandler events into a Result based on the transformation instructions specified by the `source` argument.
public TransformerHandler newTransformerHandler(Templates templates);	Generates a TransformerHandler object that can process SAX ContentHandler events into a Result based on the transformation instructions specified by the `templates` argument.

Parameter	Description
source	The Source of the transformation instructions.
templates	The compiled transformation instructions.

newXMLFilter()

Description

Creates an XMLFilter. Returns an XMLFilter object, or null if this feature is not supported. Throws `TransformerConfigurationException` if the

TemplatesHandler cannot be created. The options are described in the following table.

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()

Description

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()

Description

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()

Description

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

Description of JXTransformer

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 of JXTransformer

```
public class JXTransformer
```

`oracle.xml.jaxp.JXTransformer`

Methods of JXTransformer

Table 3–7 Summary of Methods of JXTransformer

Method	Description
JXTransformer() on page 3-25	Constructs a JXTransformer object.
clearParameters() on page 3-25	Clears all parameters set with setParameter.
getErrorListener() on page 3-26	Retrieves the error event handler in effect for the transformation
getOutputProperties() on page 3-26	Retrieves a copy of the output properties for the transformation.
getOutputProperty() on page 3-27	Returns string value of an output property that is in effect for the transformation.
getParameter() on page 3-27	Returns a parameter that was explicitly set.
getURIResolver() on page 3-27	Returns an object that will be used to resolve URIs.
setErrorListener() on page 3-28	Sets the error event listener in effect for the transformation.

Table 3–7 Summary of Methods of JXTransformer (Cont.)

Method	Description
setOutputProperties() on page 3-28	Sets output properties for the transformation
setOutputProperty() on page 3-29	Sets an output property that will be in effect for the transformation.
setParameter() on page 3-29	Adds a parameter for the transformation.
setURIResolver() on page 3-30	Sets an object that will be used to resolve URIs used in `document()`.
transform() on page 3-31	Processes the source tree to the output result.

JXTransformer()

Description

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()

Description

Clears all parameters set with setParameter.

Syntax

```
public void clearParameters();
```

getErrorListener()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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.

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

This chapter contains these sections:

oraxsl Class

Description of oraxsl

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 of oraxsl

```
public class oraxsl extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.parser.v2.oraxsl
```

Usage of oraxsl

```
java oraxsl options* source? stylesheet? result?
```

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()

Description
Class constructor.

Syntax
```
public  oraxsl();
```

main()

Description
Invokes the oraxsl driver.

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

Parameter	Description
args	Command line arguments.

XPathException Class

Description of XPathException

Indicates that an exception occurred during XPath processing.

Syntax of XPathException

```
public class XPathException extends oracle.xml.parser.v2.XSLException

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.xml.util.XMLException
                    |
                    +--oracle.xml.parser.v2.XSLException
                          |
                          +--oracle.xml.parser.v2.XPathException
```

Implemented Interfaces of XPathException

`java.io.Serializable`

Methods of XPathException

Table 4–3 Summary of Methods of XPathException

Method	Description
getErrorID() on page 4-4	Retrieves error id.
getMessage() on page 4-5	Retrieves error message.

getErrorID()

Description

Retrieves error id.

Syntax

```
public int getErrorID();
```

getMessage()

Description

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

Description of XSLException

Indicates that an exception occurred during XSL transformation.

Syntax of XSLException

```
public class XSLException extends oracle.xml.util.XMLException

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.xml.util.XMLException
                    |
                    +--oracle.xml.parser.v2.XSLException
```

Direct Subclasses of XSLException

XPathException

Implemented Interfaces of XSLException

java.io.Serializable

Methods of XSLException

XSLException()

Description

Generates a new XSL exception.

Syntax

```
public  XSLException( String mesg);
```

XSLExtensionElement Class

Description of XSLExtensionElement

Base element for extension elements

Syntax of XSLExtensionElement

```
public class XSLExtensionElement
```

oracle.xml.parser.v2.XSLExtensionElement

Methods of XSLExtensionElement

Table 4–4 Summary of Methods of XSLExtensionElement

Method	Description
XSLExtensionElement() on page 4-7	Default constructor.
getAttributeTemplateValue() on page 4-7	Retrieves an attribute value as template.
getAttributeValue() on page 4-8	Retrieves an attribute value.
getChildNodes() on page 4-8	Retrieves the childNodes of the extension elements.
processAction() on page 4-8	Executes the body of the extension elements.
processContent() on page 4-9	Processes contents of the extension element.

XSLExtensionElement()

Description

Default constructor.

Syntax

```
public  XSLExtensionElement();
```

getAttributeTemplateValue()

Description

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()

Description

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()

Description

Retrieves the childNodes of the extension elements as a nodelist.

Syntax

```
protected final java.util.Vector getChildNodes();
```

processAction()

Description

Executes the body of the extension elements.

Syntax

```
public void processAction( XSLTContext context);
```

Parameter	Description
context	XSLT Context.

processContent()

Description

Processes contents of the extension element.

Syntax

```
protected final void processContent(XSLTContext context);
```

Parameter	Description
context	XSLTContext.

XSLProcessor Class

Description of XSLProcessor

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 of XSLProcessor

```
public class XSLProcessor
```

`oracle.xml.parser.v2.XSLProcessor`

Methods of XSLProcessor

Table 4–5 Summary of Methods of XSLProcessor

Method	Description
XSLProcessor() on page 4-10	Default constructor.
getParam() on page 4-11	Retrieves the value of top-level stylesheet parameter.
newXSLStylesheet() on page 4-11	Constructs an XSLStylesheet.
processXSL() on page 4-12	Transforms input XML document.
removeParam() on page 4-15	Removes the value of a top-level stylesheet parameter.
resetParams() on page 4-15	Resets all the params set.
setBaseURL() on page 4-15	Sets base url to resolve include/import hrefs.
setEntityResolver() on page 4-15	Sets entity resolver to resolve include/import hrefs.
setErrorStream() on page 4-16	Creates an output stream for the output of warnings.
setLocale() on page 4-16	Sets the locale for error reporting.
setParam() on page 4-16	Sets the value of a top-level stylesheet parameter.
showWarnings() on page 4-17	Sets the overriding XSLOutput object.

XSLProcessor()

Description

Default constructor.

Syntax

```
public  XSLProcessor();
```

getParam()

Description

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()

Description

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()

Description

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.

Syntax	Description
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.
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.

Syntax	Description
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,
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()

Description

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()

Description

Resets all the params set. Throws an `XSLException` on error.

Syntax

```
public void resetParams();
```

setBaseURL()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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

Description of XSLStylesheet

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 of XSLStylesheet

```
public class XSLStylesheet
```

`oracle.xml.parser.v2.XSLStylesheet`

Fields of XSLStylesheet

Table 4–6 Fields of XSLStylesheet

Field	Syntax	Description
output	public oracle.xml.parser.v2.XSLOutput output	Output

Methods of XSLStylesheet

Table 4–7 Summary of Methods of XSLStylesheet

Method	Description
getDecimalFormat() on page 4-19	Returns the decimal format symbols specified in the stylesheet
getOutputEncoding() on page 4-19	Returns the value of the encoding specified in `xsl:output`
getOutputMediaType() on page 4-19	Returns the value of the media-type specified in `xsl:output`
getOutputProperties() on page 4-19	Returns the output properties specified in `xsl:output` as `java.util.Properties.`
newTransformer() on page 4-20	Returns a JAXP Transformer object that uses this stylesheet for transformation.
getContextNode() on page 4-21	Removes the value of a top-level stylesheet parameter.
getContextPosition() on page 4-22	Resets all the params set.

Table 4–7 Summary of Methods of XSLStylesheet (Cont.)

Method	Description
getContextPosition() on page 4-22	Sets the value of a top-level stylesheet parameter.

getDecimalFormat()

Description
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()

Description
Returns the value of the encoding specified in `xsl:output.`

Syntax
```
public String getOutputEncoding();
```

getOutputMediaType()

Description
Returns the value of the media-type specified in `xsl:output.`

Syntax
```
public jString getOutputMediaType();
```

getOutputProperties()

Description
Returns the output properties specified in `xsl:output` as `java.util.Properties`.

Syntax
```
public java.util.Properties getOutputProperties();
```

newTransformer()

Description

Returns a JAXP Transformer object that uses this stylesheet for transformation.

Syntax

```
public javax.xml.transform.Transformer newTransformer();
```

XSLTContext Class

Description of XSLTContext

Class for Xpath processing Context.

Syntax of XSLTContext

```
public class XSLTContext extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.parser.v2.XSLTContext
```

Methods of XSLTContext

Table 4–8 Summary of Methods of XSLTContext

Method	Description
getContextNode() on page 4-21	Returns the current context node.
getContextPosition() on page 4-22	Returns the current context node position.
getContextSize() on page 4-22	Returns the current context size.
getError() on page 4-22	Returns the XMLError instance for reporting errors.
getVariable() on page 4-22	Returns variable at the given stack offset.
reportCharacters() on page 4-22	Reports characters to the current output handler.
reportNode() on page 4-23	Reports an XMLNode to the current output handler.
setError() on page 4-23	Sets the XMLError.

getContextNode()

Description

Returns the current context node.

Syntax

```
public XMLNode getContextNode();
```

getContextPosition()

Description
Returns the current context node position.

Syntax
```
public int getContextPosition();
```

getContextSize()

Description
Returns the current context size.

Syntax
```
public int getContextSize();
```

getError()

Description
Returns the XMLError instance for reporting errors.

Syntax
```
public XMLError getError();
```

getVariable()

Description
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()

Description
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. TRUE means *disabled*, FALSE means *enabled*.

reportNode()

Description

Reports a XMLNode to the current output handler.

Syntax

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

Parameter	Description
node	Node to be output.

setError()

Description

Sets the XMLError.

Syntax

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

Parameter	Description
err	Instance of XMLError.

5

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:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

CXMLHandlerBase Class

Description of CXMLHandlerBase

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 of CXMLHandlerBase

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

`oracle.xml.comp.CXMLHandlerBase`

Implemented Interfaces of CXMLHandlerBase

`oracle.xml.parser.v2.XMLDocumentHandler`

Methods of CXMLHandlerBase

Table 5–1 Summary of Methods of CXMLHandlerBase

Method	Description
CXMLHandlerBase() on page 5-3	Creates a new CXMLHandlerBase.
cDATASection() on page 5-4	Receives notification of a CDATA Section.
characters() on page 5-4	Receives notification of Character Data inside an element.
comment() on page 5-5	Receives notification of Comment.
endDoctype() on page 5-5	Receives notification of end of the DTD.
endDocument() on page 5-5	Receives notification of end of the Document.
endElement() on page 5-6	Receives notification of end of the Element.
endPrefixMapping() on page 5-6	Receives notification of end of scope of Prefix Mapping.
getCXMLContext() on page 5-6	Returns the CXML Context used for compression.
getProperty() on page 5-7	Looks up and returns the value of a property.
ignorableWhitespace() on page 5-7	Receives notification of ignorable whitespace in element content.
processingInstruction() on page 5-7	Receives notification of a processing instruction.

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

Method	Description
setDoctype() on page 5-8	Sets DTD to receive notification of that DTD.
setDocumentLocator() on page 5-8	Sets Locator to receive that Locator object for the document event.
setError() on page 5-9	Sets XMLError handler to receive notification of that XMLError handler.
setProperty() on page 5-9	Sets the value of a property.
setTextDecl() on page 5-10	Sets Text XML Declaration to receive notification of that Text XML Declaration.
setXMLDecl() on page 5-10	Sets XML Declaration to receive notification of that XML Declaration.
setXMLSchema() on page 5-11	Sets XMLSchema to receive notification of that XMLSchema object.
skippedEntity() on page 5-11	Receives notification of a skipped entity.
startDocument() on page 5-12	Receives notification of the beginning of the document.
startElement() on page 5-12	Receives notification of the beginning of an element.
startPrefixMapping() on page 5-12	Receives notification of the beginning of the scope of prefix URI mapping.

CXMLHandlerBase()

Description

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()

Description

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()

Description

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.

Parameter	Description
len	The number of characters to use from the character array.

comment()

Description

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()

Description

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()

Description

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()

Description

Receives notification of the end of the element.

Syntax

```
public void endElement( oracle.xml.parser.v2.NSName elem);
```

Parameter	Description
elem	The element.

endPrefixMapping()

Description

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()

Description

Returns the CXML Context used for compression.

Syntax

```
public oracle.xml.comp.CXMLContext getCXMLContext();
```

getProperty()

Description

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()

Description

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.
len	The number of characters to read from the array.

processingInstruction()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

```
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()

Description

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()

Description

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()

Description

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.

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

CXMLParser Class

Description of CXMLParser

This class implements the re-generation of XML document from the compressed stream by generating the SAX events from them.

Syntax of CXMLParser

```
public class CXMLParser
```

```
oracle.xml.comp.CXMLParser
```

Methods of CXMLParser

Table 5–2 Summary of Methods of CXMLHandlerBase

Method	Description
startPrefixMapping() on page 5-12	Creates a new XML Parser object for reading the compressed stream.
CXMLParser() on page 5-14	Retrieves content handler.
getContentHandler() on page 5-15	Retrieves the current error handler.
getErrorHandler() on page 5-15	Parses the compressed stream and generates the SAX events.
parse() on page 5-15	Parses the compressed stream and generates the SAX events.
setContentHandler() on page 5-16	Registers a content event handler.
setErrorHandler() on page 5-16	Registers an error event handler.

CXMLParser()

Description

Creates a new XML Parser object for reading the compressed stream.

Syntax

```
public  CXMLParser();
```

getContentHandler()

Description

Retrieves content handler; returns `NULL` if the handler has not been registered.

Syntax

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

getErrorHandler()

Description

Retrieves the current error handler; returns `NULL` if the handler has not been registered.

Syntax

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

parse()

Description

Parses the compressed stream and generates the SAX events. Throws the following exceptions:

- `SAXException` - any SAX exception
- `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()

Description
Registers a content event handler.

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

Parameter	Description
handler	The content handler

setErrorHandler()

Description
Registers an error event handler.

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

Parameter	Description
handler	The error handler

6

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:

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

XMLSchema Class

Description of XMLSchema

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 of XMLSchema

```
public class XMLSchemaNode extends oracle.xml.parser.schema.XSDNode

oracle.xml.parser.schema.XSDNode
  |
  +--oracle.xml.parser.schema.XMLSchemaNode
```

Methods of XMLSchema

Table 6–1 Summary of Methods of XML Schema

Constructor	Description
XMLSchema() on page 6-2	XMLSchema constructor.
getAllTargetNS() on page 6-3	Returns all the Target Name space defined in the schema.
getSchemaByTargetNS() on page 6-3	Returns schemaNode for the given namespace.
getSchemaTargetNS() on page 6-3	Returns the top level schema's target Namespace.
getXMLSchemaNodeTable() on page 6-4	Returns XMLSchemaNode table.
getXMLSchemaURLS() on page 6-4	Returns XMLSchema URLs.
printSchema() on page 6-4	Print schema information.

XMLSchema()

Description

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()

Description
Returns all the Target Name space defined in the schema.

Syntax
```
public java.lang.String[] getAllTargetNS();
```

getSchemaByTargetNS()

Description
Returns schemaNode for the given namespace.

Syntax
```
public XMLSchemaNode getSchemaByTargetNS( String namespace);
```

Parameter	Description
namespace	Target namespace of the required schema.

getSchemaTargetNS()

Description
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()

Description

Returns XMLSchemaNode table as a hashtable.

Syntax

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

getXMLSchemaURLS()

Description

Returns XMLSchema URLs as an array.

Syntax

```
public java.lang.String[] getXMLSchemaURLS();
```

printSchema()

Description

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

Description of XMLSchemaNode

This class contains sets of top level Schema components in a target namespace.

Syntax of XMLSchemaNode

```
public class XMLSchemaNode extends oracle.xml.parser.schema.XSDNode

oracle.xml.parser.schema.XSDNode
  |
  +--oracle.xml.parser.schema.XMLSchemaNode
```

Methods of XMLSchemaNode

Table 6–2 Summary of Methods of XMLSchemaNode

Method	Description
XMLSchemaNode() on page 6-5	XMLSchema constructor.
getAttributeDeclarations() on page 6-6	Returns all the top level attributes in the schema.
getComplexTypeSet() on page 6-6	Returns all the top level complex type elements in the schema.
getComplexTypeTable() on page 6-6	Returns the complex type definitions.
getElementSet() on page 6-6	Returns all the top level elements in the schema.
getSimpleTypeSet() on page 6-6	Returns all the top level simpleType elements in the schema.
getSimpleTypeTable() on page 6-7	Returns the simple type definitions.
getTargetNS() on page 6-7	Returns targetNS of the schema.
getTypeDefinitionTable() on page 6-7	Returns the type definitions.

XMLSchemaNode()

Description

XMLSchema constructor.

Syntax

```
public  XMLSchemaNode();
```

getAttributeDeclarations()

Description

Returns all the top level attributes in the schema as an array.

Syntax

```
public XSDAttribute getAttributeDeclarations();
```

getComplexTypeSet()

Description

Returns all the top level complex type elements in the schema as an array.

Syntax

```
public XSDNode getComplexTypeSet();
```

getComplexTypeTable()

Description

Returns the complex type definitions as a hashtable.

Syntax

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

getElementSet()

Description

Returns all the top level elements in the schema as an array.

Syntax

```
public XSDNode getElementSet();
```

getSimpleTypeSet()

Description

Returns all the top level simpleType elements in the schema as an array.

Syntax

```
public XSDNode getSimpleTypeSet();
```

getSimpleTypeTable()

Description

Returns the simple type definitions, in a hashtable.

Syntax

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

getTargetNS()

Description

Returns targetNS of the schema. Overrides `XSDNode.getTargetNS()` in class `XSDNode.`

Syntax

```
public String getTargetNS();
```

getTypeDefinitionTable()

Description

Returns the type definitions as a hashtable.

Syntax

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

XSDAttribute Class

Description of XSDAttribute

This class represents Schema Attribute declaration.

Syntax of XSDAttribute

```
public class XSDAttribute extends oracle.xml.parser.schema.XSDNode

oracle.xml.parser.schema.XSDNode
  |
  +--oracle.xml.parser.schema.XSDAttribute
```

Methods of XSDAttribute

Table 6–3 Summary of Methods of XSDAttribute

Method	Description
getDefaultVal() on page 6-8	Returns the value of 'default' attributes in case of element, and the value of 'value' attributes based on 'use' attribute.
getFixedVal() on page 6-9	Returns the value of 'fixed' attribute in case of element, and the value of 'value' attribute based on 'use' attribute.
getName() on page 6-9	Returns the name of the node.
getRefLocalname() on page 6-9	Returns he local name of the resolved 'ref' attribute.
getRefNamespace() on page 6-9	Returns the namespace of the resolved 'ref' attribute.
getRefState() on page 6-10	Returns refState.
getTargetNS() on page 6-10	Returns target namespace.
getType() on page 6-10	Returns the node type.
isRequired() on page 6-10	Checks if the attribute is required.

getDefaultVal()

Description

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()

Description

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()

Description

Returns the name of the node. Overrides `XSDNode.getName()` in class `XSDNode.`

Syntax

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

getRefLocalname()

Description

Returns the refLocal name of the resolved 'ref' attribute

Syntax

```
public String getRefLocalname();
```

getRefNamespace()

Description

Returns the RefNamespace of the resolved 'ref' attribute.

Syntax

```
public String getRefNamespace();
```

getRefState()

Description

Returns refState value. The return value is one of the following: `TYPE_UNRESOLVED`, `TYPE_RESOLVED`, `REF_UNRESOLVED`, `REF_RESOLVED`.

Syntax

```
public int getRefState();
```

getTargetNS()

Description

Returns target namespace. `XSDNode.getTargetNS()` in class `XSDNode`.

Syntax

```
public String getTargetNS();
```

getType()

Description

Returns the node type which is either simpleTypeor complexType.

Syntax

```
public XSDNode getType();
```

isRequired()

Description

Checks if the attribute is required.

Syntax

```
public boolean isRequired();
```

XSDBuilder Class

Description of XSDBuilder

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 of XSDBuilder

```
public class XSDBuilder
```

Methods of XSDBuilder

Table 6–4 Summary of Methods of XSDBuilder

Method	Description
XSDBuilder() on page 6-11	Class constructor.
build() on page 6-12	Builds an XMLSchema object or document.
getObject() on page 6-13	Returns XML schema object.
setEntityResolver() on page 6-14	Sets an EntityResolver for resolving imports/include.
setError() on page 6-14	Sets XMLError object.
setLocale() on page 6-14	Sets locale for error reporting.

XSDBuilder()

Description

XSDBuilder constructor.

Syntax

```
public XSDBuilder() throws XSDException;
```

build()

Description

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.

Syntax	Description
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 any relative refs; used 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()

Description

Returns XML schema object.

Syntax

```
public Object getObject();
```

setEntityResolver()

Description

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()

Description

Sets XMLError object.

Syntax

public void setError(XMLError er);

Parameter	Description
er	XMLError object

setLocale()

Description

Sets locale for error reporting.

Syntax

```
public void setLocale(L ocale locale);
```

Parameter	Description
locale	Locale object

XSDComplexType Class

Description of XSDComplexType

This class represents the Schema ComplexType definition.

Syntax of XSDComplexType

```
public class XSDComplexType extends oracle.xml.parser.schema.XSDNode

oracle.xml.parser.schema.XSDNode
  |
  +--oracle.xml.parser.schema.XSDComplexType
```

Methods XSDComplexType

Table 6–5 Summary of Methods of XSDComplexType

Method	Description
getAttributeDeclarations() on page 6-16	Returns attribute declarations of this complex type.
getAttributeWildcard() on page 6-16	Returns attribute wildcard of this complex type.
getBaseType() on page 6-16	Returns the base type of this complextype.
getContent() on page 6-16	Returns content of XSDComplexType.
getDerivationMethod() on page 6-17	Returns information on how was this type derived from its parent type.
getElementSet() on page 6-17	Returns an array of all the local elements inside a complexType element.
getGroup() on page 6-17	Returns the attribute group or the child & attribute group.
getRefLocalname() on page 6-17	Returns the local name of resolved 'base' attr.
getTypeGroup() on page 6-17	Returns type group of Complex Type.
init() on page 6-18	Initializes complex type.
isAbstract() on page 6-18	Returns TRUE if the Complex Type is abstract.

getAttributeDeclarations()

Description
Returns attribute declarations of this complex type; does not include wild card array of attribute declarations.

Syntax
```
public XSDAttribute getAttributeDeclarations();
```

getAttributeWildcard()

Description
Returns attribute wildcard of this complex type.

Syntax
```
public oracle.xml.parser.schema.XSDAny getAttributeWildcard();
```

Returns
The attribute wildcard if has one

getBaseType()

Description
Returns the base type of this complextype.

Syntax
```
public XSDNode getBaseType();
```

getContent()

Description
Returns content of XSDComplexType.

Syntax
```
public int getContent();
```

getDerivationMethod()

Description

Returns information on how was this type derived from its parent type. One of `EXTENSION_DERIVATION` or `RESTRICTION_DERIVATION`.

Syntax

```
public short getDerivationMethod();
```

getElementSet()

Description

Returns an array of all the local elements inside a complexType element.

Syntax

```
public XSDNode getElementSet();
```

getGroup()

Description

Returns the attribute group or the child & attribute group.

Syntax

```
public XSDGroup getGroup();
```

getRefLocalname()

Description

Returns the local name of resolved 'base' attr.

Syntax

```
public java.lang.String getRefLocalname();
```

getTypeGroup()

Description

Returns type group of Complex Type.

Syntax

```
public XSDGroup getTypeGroup();
```

init()

Description

Initializes complex type.

Syntax

```
public static void init();
```

isAbstract()

Description

Returns TRUE if the Complex Type is abstract.

Syntax

```
public boolean isAbstract();
```

XSDConstrainingFacet Class

Description of XSDConstrainingFacet

This class represents Schema Constraining facet for Data Type.

Syntax of XSDConstrainingFacet

```
public class XSDConstrainingFacet extends java.lang.Object implements
oracle.xml.parser.schema.XSDTypeConstants

java.lang.Object
  |
  +--oracle.xml.parser.schema.XSDConstrainingFacet
```

Implemented Interfaces of XSDConstrainingFacet

`XSDTypeConstants`

Methods of XSDConstrainingFacet

Table 6–6 Summary of Methods of XSDConstrainingFacet

Method	Description
getFacetId() on page 6-19	Returns the facet id.
getLexicalEnumeration() on page 6-20	Returns the LEXICAL enumeration of the constraining facet.
getLexicalValue() on page 6-20	Returns the LEXICAL value of the constraining facet.
getName() on page 6-20	Returns the name of the constraining facet.
isFixed() on page 6-20	Checks whether facet is fixed and cannot be changed.
setFixed() on page 6-21	Sets if the facet is fixed and cannot be changed.
validateFacet() on page 6-21	Validates the value against the constraint facet.

getFacetId()

Description

Returns the facet id.

Syntax

```
public int getFacetId();
```

getLexicalEnumeration()

Description

Returns the LEXICAL enumeration of the constraining facet.

Syntax

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

getLexicalValue()

Description

Returns the LEXICAL value of a constraining facet.

Syntax

```
public String getLexicalValue();
```

getName()

Description

Returns the name of the constraining facet.

Syntax

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

isFixed()

Description

Checks whether facet is fixed and cannot be changed. Returns `TRUE` for fixed, `FALSE` otherwise.

Syntax

```
public boolean isFixed();
```

setFixed()

Description

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()

Description

Validates the value against the constraint facet.

Syntax

```
public void validateFacet( XSDDataValue value);
```

Parameter	Description
value	Value being validated.

XSDDataValue Class

Description of XSDDataValue

This class represents data values for Schema Simple Type.

Syntax of XSDDataValue

```
public class XSDDataValue extends java.lang.Object implements
oracle.xml.parser.schema.XSDTypeConstants

java.lang.Object
  |
  +--oracle.xml.parser.schema.XSDDataValue
```

Implemented Interfaces of XSDDataValue

XSDTypeConstants

Methods of XSDDataValue

Table 6–7 Summary of Methods of XSDDataValue

Method	Description
compareTo() on page 6-22	Compares two values returns -1 for smaller, 0 for equal, and 1 for greater.
getLength() on page 6-23	Returns the length of STRING/BINARY value.
getLexicalValue() on page 6-23	Returns LEXICAL value from the XSDDataValue class.
getPrecision() on page 6-23	Returns the precision of decimal value.
getScale() on page 6-23	Returns the scale of decimal value.

compareTo()

Description

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()

Description

Gets the length of STRING / BINARY value. Throws `XSDException` if the data value is not of String / Binary type.

Syntax

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

getLexicalValue()

Description

Returns LEXICAL value from the XSDDataValue class.

Syntax

```
public String getLexicalValue();
```

getPrecision()

Description

Returns the precision of decimal value. Throws `XSDException` if the data values are not decimal type.

Syntax

```
public int getPrecision();
```

getScale()

Description

Returns the int value of a decimal scale. Throws `XSDException` if the data value is not decimal type.

Syntax

```
public int getScale();
```

XSDElement Class

Description of XSDElement

The XSDElement class represents the Schema element declaration.

Syntax of XSDElement

```
public class XSDElement
```

`oracle.xml.parser.schema.XSDElement`

Methods of XSDElement

Table 6–8 Summary of Methods of XSDElement

Method	Description
findEquivClass() on page 6-25	Finds the equivalent class corresponding to this class.
getDefaultVal() on page 6-25	Returns the value of 'default' attr in case of element, and the value of 'value' attr based on 'use' attribute.
getEquivClassRef() on page 6-25	Returns the local name of the resolved equivalent class.
getFixedVal() on page 6-26	Returns the value of 'fixed' attr in case of element, and the value of 'value' attr based on 'use' attribute.
getIdentities() on page 6-26	Returns the set of identities.
getMaxOccurs() on page 6-26	Returns the maxOccurs.
getMinOccurs() on page 6-26	Returns the minOccurs.
getName() on page 6-27	Returns Name.
getRefLocalname() on page 6-27	Returns the local name of the resolved 'ref' attribute.
getRefNamespace() on page 6-27	Returns the namespace of the resolved 'ref' attribute.
getRefState() on page 6-27	Returns refState.
getSubstitutionGroup() on page 6-28	Returns the substitutionGroup.
getTargetNS() on page 6-28	Sets target namespace.
getType() on page 6-28	Sets the node type.
isAbstract() on page 6-28	Returns TRUE if element abstract.
isNullable() on page 6-28	Returns TRUE if element nullable.

Table 6–8 Summary of Methods of XSDElement (Cont.)

Method	Description
setMaxOccurs() on page 6-29	Sets the maxOccurs.
setMinOccurs() on page 6-29	Sets the minOccurs.

findEquivClass()

Description

Finds the equivalent class corresponding to this class.

Syntax

```
public XSDElement findEquivClass( String ns,
                                  String nm);
```

Parameter	Description
ns	Namespace.
rm	Name.

getDefaultVal()

Description

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()

Description

Returns the local name of the resolved equivalent class.

Syntax

```
public String getEquivClassRef();
```

getFixedVal()

Description

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()

Description

Returns the set of identities, as an array.

Syntax

```
public XSDIdentity getIdentities();
```

getMaxOccurs()

Description

Returns the maxOccurs

Syntax

```
public int getMaxOccurs();
```

getMinOccurs()

Description

Returns the minOccurs.

Syntax

```
public int getMinOccurs();
```

getName()

Description

Returns the name of the node.

Syntax

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

getRefLocalname()

Description

Returns the local name of the resolved 'ref' attribute

Syntax

```
public String getRefLocalname();
```

getRefNamespace()

Description

Returns the namespace of the resolved 'ref' attribute

Syntax

```
public String getRefNamespace();
```

getRefState()

Description

Returns refState. The return value is one of the following: `TYPE_UNRESOLVED`, `TYPE_RESOLVED`, `REF_UNRESOLVED`, `REF_RESOLVED`.

Syntax

```
public int getRefState();
```

getSubstitutionGroup()

Description

Returns the substitutionGroup

Syntax

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

getTargetNS()

Description

Returns target namespace.

Syntax

```
public java.lang.String getTargetNS();
```

getType()

Description

Returns the node type, either simpleType or complexType.

Syntax

```
public XSDNode getType();
```

isAbstract()

Description

Returns TRUE if this element is abstract.

Syntax

```
public boolean isAbstract();
```

isNullable()

Description

Returns TRUE if this element is nullable.

Syntax

```
public boolean isNullable();
```

setMaxOccurs()

Description

Sets the maxOccurs.

Syntax

```
public void setMaxOccurs( int max);
```

Parameter	Description
maxOccurs	value

setMinOccurs()

Description

Sets the minOccurs.

Syntax

```
public void setMinOccurs( int min);
```

Parameter	Description
minOccurs	value

XSDException

Description of XSDException

Indicates that an exception occurred during XMLSchema validation.

Syntax of XSDException

```
java.lang.Object
   |
   +---java.lang.Throwable
           |
           +---java.lang.Exception
                   |
                   +---oracle.xml.parser.schema.XSDException

public class XSDException extends Exception
```

getMessage()

Description

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

Description of XSDGroup

The XSDGroup class represents the Schema group definition.

Syntax of XSDGroup

```
public class XSDGroup
```

oracle.xml.parser.schema.XSDGroup

Methods of XSDGroup

Table 6–9 Summary of Methods of XSDIdentity

Method	Description
getMaxOccurs() on page 6-31	Returns the maxOccurs.
getMinOccurs() on page 6-31	Returns the minOccurs.
getNodeVector() on page 6-32	Returns the particles of the group stored in nodeVector.
getOrder() on page 6-32	Returns the composite type - `ALL`, `SEQUENCE`, `CHOICE`.
setMaxOccurs() on page 6-32	Sets maxOccurs.
setMinOccurs() on page 6-32	Sets minOccurs.

getMaxOccurs()

Description

Returns the maxOccurs.

Syntax

```
public int getMaxOccurs();
```

getMinOccurs()

Description

Returns the minOccurs.

Syntax

```
public int getMinOccurs();
```

getNodeVector()

Description

Returns the particles of the group stored in nodeVector.

Syntax

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

getOrder()

Description

Returns the composite type - `ALL`, `SEQUENCE`, or `CHOICE`

Syntax

```
public int getOrder();
```

setMaxOccurs()

Description

Sets maxOccurs.

Syntax

```
public void setMaxOccurs( int max);
```

Parameter	Description
maxOccurs	value

setMinOccurs()

Description

Sets the minOccurs.

Syntax

```
public void setMinOccurs( int min);
```

Parameter	Description
minOccurs	value

XSDIdentity Class

Description of XSDIdentity

This class represents Schema Identity-Constraint definition: `Unique`, `Key`, and `Keyref`.

Syntax of XSDIdentity

```
public class XSDIdentity extends oracle.xml.parser.schema.XSDNode

oracle.xml.parser.schema.XSDNode
  |
  +--oracle.xml.parser.schema.XSDIdentity
```

Methods of XSDIdentity

Table 6–10 Summary of Methods of XSDIdentity

Method	Description
getFields() on page 6-34	Returns the fields.
getNodeType() on page 6-34	Returns the node type.
getRefer() on page 6-35	Returns the reference key.
getSelector() on page 6-35	Returns the selector.

getFields()

Description

Returns the fields.

Syntax

```
public java.lang.String[] getFields();
```

getNodeType()

Description

Returns the node Type. Overrides `XSDNode.getNodeType()` in class `XSDNode.`

Syntax

```
public int getNodeType();
```

getRefer()

Description

Returns the referenced key

Syntax

```
public String getRefer();
```

getSelector()

Description

Returns the selector.

Syntax

```
public String getSelector();
```

XSDNode Class

Description of XSDNode

Root class for most of XSD classes. Contains fields and methods corresponding to XMLSchema definition attributes.

Syntax of XSDNode

`public class XSDNode`

`oracle.xml.parser.schema.XSDNode`

Direct Subclasses of XSDNode

`XMLSchema, XMLSchemaNode, XSDAttribute, XSDComplexType, XSDIdentity`

Methods of XSDNode

Table 6–11 Summary of Methods of XSDNode

Method	Description
getName() on page 6-36	Returns the name of the node.
getNamespaceURI() on page 6-37	Returns namespace uri for psv.
getNodeType() on page 6-37	Returns the type of XSDNode.
getTargetNS() on page 6-37	Returns target namespace.
isNodeType() on page 6-37	Checks if the node is of the give type.

getName()

Description

Returns the name of the node.

Syntax

```
public java.lang.String getName();
```

getNamespaceURI()

Description

Returns namespace uri for psv.

Syntax

```
public String getNamespaceURI()
```

getNodeType()

Description

Returns the type of XSDNode.

Syntax

```
public int getNodeType();
```

getTargetNS()

Description

Returns target namespace.

Syntax

```
public java.lang.String getTargetNS();
```

isNodeType()

Description

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

Description of XSDSimpleType

This class represents Schema Simple Type definition.

Syntax of XSDSimpleType

```
public class XSDSimpleType implements oracle.xml.parser.schema.XSDTypeConstants
```

`oracle.xml.parser.schema.XSDSimpleType`

Implemented Interfaces of XSDSimpleType

`XSDTypeConstants`

Methods of XSDSimpleType

Table 6–12 Summary of Methods of XSDSimpleType

Method	Description
XSDSimpleType() on page 6-39	Class constructor.
derivedFrom() on page 6-39	Derive a type from the given base type.
getBase() on page 6-40	Returns the base type of this type.
getBasicType() on page 6-40	Gets the basic type from which this type was derived.
getBuiltInDatatypes() on page 6-40	Gets a built-in datatype.
getFacets() on page 6-41	Get the facets.
getMaxOccurs() on page 6-41	Get the value of maxOccurs.
getMinOccurs() on page 6-41	Get the value of minOccurs.
getVariety() on page 6-41	Get the variety of the type.
isAbstract() on page 6-41	Returns TRUE if this Type is abstract.
setFacet() on page 6-42	Sets a facet for the datatype (Internal private API).
setMaxOccurs() on page 6-42	Sets the value of maxOccurs.
setMinOccurs() on page 6-42	Sets the value of minOccurs.
setSource() on page 6-43	Sets the base type of the datatype, or in case of aggregate types sets the type of the component of the aggregate type.

Table 6–12 Summary of Methods of XSDSimpleType (Cont.)

Method	Description
validateValue() on page 6-43	Validates the string value with the facets defined for this type.

XSDSimpleType()

Description

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()

Description

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()

Description
Retrieves the base type.

Syntax
```
public XSDSimpleType getBase();
```

getBasicType()

Description
Retrieves the basic type from which this type was derived.

Syntax
```
public int getBasicType();
```

getBuiltInDatatypes()

Description
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()

Description

Retrieves the facets.

Syntax

```
public XSDConstrainingFacet getFacets();
```

getMaxOccurs()

Description

Retrieves value of maxOccurs.

Syntax

```
public int getMaxOccurs();
```

getMinOccurs()

Description

Retrieves the value of minOccurs.

Syntax

```
public int getMinOccurs();
```

getVariety()

Description

Retrieves the variety of the type.

Syntax

```
public java.lang.String getVariety();
```

isAbstract()

Description

Returns TRUE if this type is abstract, FALSE otherwise.

Syntax

```
public boolean isAbstract();
```

setFacet()

Description

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()

Description

Set the value of maxOccurs

Syntax

```
public void setMaxOccurs(int max);
```

Parameter	Description
max	Number of maximum occurrences.

setMinOccurs()

Description

Set the value of minOccurs.

Syntax

```
public void setMinOccurs( int min);
```

Parameter	Description
min	Number of minimum occurrences.

setSource()

Description

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()

Description

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

Description of XSDTypeConstantValues

This interface defines constants for the W3C Schema Processor.

Syntax of XSDTypeConstantValues

```
public interface XSDTypeConstants
```

Constants Defined in XSDConstantValues

Table 6–13 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

Table 6–13 Constants Defined in XSDConstantValues (Cont.)

Constant	Definition
_default	public static final String _default
_derivedBy	public static final String _derivedBy
_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

Table 6–13 Constants Defined in XSDConstantValues (Cont.)

Constant	Definition
_mixed	public static final String _mixed
_nall	public static final String _nall
_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

Table 6–13 Constants Defined in XSDConstantValues (Cont.)

Constant	Definition
_schema	public static final String _schema
_schemaLocation	public static final String _schemaLocation
_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

Table 6–13 Constants Defined in XSDConstantValues (Cont.)

Constant	**Definition**
ANY_ATTRIBUTE	public static final int ANY_ATTRIBUTE
ANY_NODE	public static final String ANY_NODE
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

Table 6–13 Constants Defined in XSDConstantValues (Cont.)

Constant	Definition
IDENTITY_KEYREF	public static final int IDENTITY_KEYREF
IDENTITY_UNIQUE	public static final int IDENTITY_UNIQUE
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

Table 6–13 Constants Defined in XSDConstantValues (Cont.)

Constant	Definition
noDefinition	public static final int noDefinition
NOT_DONE	public static final int NOT_DONE
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

Table 6–13 Constants Defined in XSDConstantValues (Cont.)

Constant	Definition
XSDAUG2000NS	public static final String XSDAUG2000NS
XSDDATATYPENS	public static final String XSDDATATYPENS
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

Description of XSDValidator

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 yjr 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 of XSDValidator

```
public class XSDValidator
```

`oracle.xml.parser.schema.XSDValidator`

Methods of XSDValidator

Table 6–14 Summary of Methods of XSDValidator

Method	Description
XSDValidator() on page 6-53	Class constructor.
characters() on page 6-53	Propagates notification of character data inside an element.
endElement() on page 6-54	Receives notification of the end of an element.
setDocumentLocator() on page 6-54	Propagates Locator object for document events.
setError() on page 6-55	Sets an XMLError object as current err.
setXMLProperties() on page 6-55	Sets XML Properties for runtime properties.
setXMLProperty() on page 6-55	Sets a property.
startElement() on page 6-56	Receive notification of a beginning of the element.

XSDValidator()

Description

XSDValidator constructor.

Syntax

```
public  XSDValidator();
```

characters()

Description

Propagate notification of character data inside an element. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception. 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()

Description

Receive notification of the end of an element. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception.

Syntax

```
public void endElement( String namespaceURI,
                        String localName,
                        String qName);
```

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 XML 1.0 name (with prefix), or the empty string if qualified names are not available.

setDocumentLocator()

Description

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()

Description

Sets an XMLError object as current err. 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	XMLError object.

setXMLProperties()

Description

Sets XML Properties for runtime properties.

Syntax

```
public void setXMLProperties( XMLProperties xmlProp);
```

Parameter	Description
xmlProp	XML properties.

setXMLProperty()

Description

Sets and returns a property. 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 Object setXMLProperty( java.lang.String name,
                              java.lang.Object value);
```

Parameter	Description
name	name of the property
value	value of the property

startElement()

Description

Receive notification of the beginning of an element. Throws `org.xml.sax.SAXException`, which can be any SAX exception, possibly wrapping another exception. See also: `endElement()`, `org.xml.sax.Attributes.`

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), 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.

7 XML Class Generation for Java

The XML Class Generator for Java is contained within the oracle.xml.classgen package.

This chapter describes API's in the following classes:

- CGDocument Class
- CGNode Class
- CGXSDElement Class
- DTDClassGenerator Class
- InvalidContentException Class
- oracg Class
- SchemaClassGenerator Class

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

CGDocument Class

Description of CGDocument

This class serves as the base document class for the DTD class Generator generated classes.

Syntax of CGDocument

```
public abstract class CGDocument extends oracle.xml.classgen.CGNode implements
java.io.Externalizable

oracle.xml.classgen.CGNode
  |
  +--oracle.xml.classgen.CGDocument
```

Implemented Interfaces of CGDocument

`java.io.Externalizable, java.io.Serializable`

Methods of CGDocument

Table 7–1 Summary of Methods of CGDocument

Method	Description
CGDocument() on page 7-2	Constructor for the root element of the DTD.
print() on page 7-3	Prints the constructed XML document.
readExternal() on page 7-3	Reads the compressed stream and creates the object corresponding to the root element.

CGDocument()

Description

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()

Description

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()

Description

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

Description of CGNode

This class serves as the base class for the classes corresponding to the nodes of XML document generated by the DTD class generator.

Syntax of CGNode

```
public abstract class CGNode
```

`oracle.xml.classgen.CGNode`

Direct Subclasses of CGNode

`CGDocument`

Fields of CGNode

Table 7–2 Fields of CGNode

Field	Syntax	Description
isValidating	protected boolean isValidating	Boolean to indicate the validating mode

Methods of CGNode

Table 7–3 Summary of Methods of CGNode

Method	Description
CGNode() on page 7-6	Constructor for the Elements of the DOM Tree.
addCDATASection() on page 7-7	Adds CDATA Section to the Element.
addData() on page 7-7	Adds PCDATA to the element node.
addNode() on page 7-7	Adds a node as a child to the element.
deleteData() on page 7-8	Deletes PCDATA from an element node.
getAttribute() on page 7-8	Retrieves the value of the attribute.
getCGDocument() on page 7-9	Retrieves the base document.
getData() on page 7-9	Retrieves the PCDATA of the element.

Table 7–3 Summary of Methods of CGNode (Cont.)

Method	Description
getDTDNode() on page 7-9	Retrieves the static DTD from the base document.
getElementNode() on page 7-9	Retrieves the XMLElement node corresponding to this CGNode.
getNode() on page 7-10	Retrieves the CGNode which is one of the children of the element corresponding to this node whose name matches the input string.
readExternal() on page 7-10	Reads the compressed stream and instantiates the corresponding node.
setAttribute() on page 7-10	Sets the value of the attribute.
setDocument() on page 7-11	Sets the base document.
setElementNode() on page 7-11	Sets the XMLElement node corresponding to this CGNode.
storeID() on page 7-12	Stores this value of ID identifier.
storeIDREF() on page 7-12	Stores this value for an IDREF identifier.
validateContent() on page 7-12	Checks if the content of the element is valid according to the Content Model specified in DTD.
validEntity() on page 7-13	Checks if the ENTITY identifier is valid.
validID() on page 7-13	Checks if the ID identifier is valid.
validNMTOKEN() on page 7-13	Checks if the NMTOKEN identifier is valid.
writeExternal() on page 7-14	Writes the compressed stream corresponding to this node.

CGNode()

Description

Constructor for the Elements of the DOM Tree.

Syntax

```
protected  CGNode( String elementName);
```

Parameter	Description
elementName	Name of the element.

addCDATASection()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Returns the value of the attribute.

Syntax

```
protected String getAttribute( String attName);
```

Parameter	Description
attName	Name of the attribute.

getCGDocument()

Description

Gets the base document (root Element).

Syntax

```
protected CGDocument getCGDocument();
```

getData()

Description

Gets the PCDATA of the Element. Throws `InvalidContentException` if the data is not present.

Syntax

```
protected String getData();
```

getDTDNode()

Description

Retrieves the static DTD from the base CGDocument.

Syntax

```
protected abstract oracle.xml.parser.v2.DTD getDTDNode();
```

getElementNode()

Description

Retrieves the XMLElement node corresponding to this CGNode.

Syntax

```
protected oracle.xml.parser.v2.XMLElement getElementNode();
```

getNode()

Description

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()

Description

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()

Description

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()

Description

Sets the base document (root element).

Syntax

```
public void setDocument( CGDocument d);
```

Parameter	Description
d	Base CGDocument.

setElementNode()

Description

Sets the XMLElement node corresponding to this CGNode.

Syntax

```
protected void setElementNode( oracle.xml.parser.v2.XMLElement node);
```

Parameter	Description
node	The XMLElement.

storeID()

Description

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()

Description

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()

Description

Checks if the content of the element is valid according to the Content Model specified in DTD.

Syntax

```
protected void validateContent();
```

validEntity()

Description

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()

Description

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()

Description

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()

Description

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

Description of CGXSDElement

This class serves as the base class for the all the generated classes corresponding to the XML Schema generated by Schema Class Generator

Syntax of CGXSDElement

```
public abstract class CGXSDElement extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.classgen.CGXSDElement
```

Fields of CGXSDElement

Table 7–4 Fields of CGXSDElement

Field	Syntax	Description
type	protected java.lang.Object type	Type information of a node

Methods of CGXSDElement

Table 7–5 Summary of Methods of CGXSDElement

Method	Description
CGXSDElement() on page 7-16	Default constructor.
addAttribute() on page 7-16	Adds the attribute of a given node to the hashtable.
addElement() on page 7-16	Adds the local elements of an element node to the vector corresponded to the elements.
getAttributes() on page 7-17	Returns the attributes as a hashtable of attribute names and values.
getChildElements() on page 7-17	Retrieves the vector of all local elements.
getNodeValue() on page 7-17	Returns the value of the node.
print() on page 7-17	Prints an element node.
printAttributes() on page 7-18	Prints an attribute node.

Table 7–5 Summary of Methods of CGXSDElement (Cont.)

Method	Description
setNodeValue() on page 7-18	Sets the node value of an element.

CGXSDElement()

Description

Default constructor.

Syntax

```
public  CGXSDElement();
```

addAttribute()

Description

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()

Description

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()

Description

Returns the attributes as a hashtable of attribute names and values.

Syntax

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

getChildElements()

Description

Retrieves the vector of all local elements.

Syntax

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

getNodeValue()

Description

Returns the value of the node.

Syntax

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

print()

Description

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()

Description

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()

Description

Sets the node value of an element.

Syntax

```
protected void setNodeValue( String value);
```

Parameter	Description
value	The node vale.

DTDClassGenerator Class

Description of DTDClassGenerator

Generates the data binding classes corresponding to a DTD or an XML file based on a DTD.

Syntax of DTDClassGenerator

```
public class DTDClassGenerator extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.classgen.DTDClassGenerator
```

Methods of DTDClassGenerator

Table 7–6 Summary of Methods of DTDClassGenerator

Method	Description
DTDClassGenerator() on page 7-19	Default constructor for DTDClassGenerator.
generate() on page 7-20	Traverses the DTD with element `doctype` as root and generates Java classes.
setGenerateComments() on page 7-20	Sets the switch to determine whether to generate java doc comments for the generated classes.
setJavaPackage() on page 7-20	Sets the package for the classes generated.
setOutputDirectory() on page 7-21	Sets the output directory where the java source code for the DTD is generated.
setSerializationMode() on page 7-21	Sets the switch to determine if the DTD should be saved as a serialized object or as text file.
setValidationMode() on page 7-22	Sets the switch to determine whether the classes generated should validate the XML document.

DTDClassGenerator()

Description

Default constructor for DTDClassGenerator.

Syntax

```
public  DTDClassGenerator();
```

generate()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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

Description of InvalidContentException

Defines the Exception thrown by DTD ClassGenerator and Schema Class Generator.

Syntax of InvalidContentException

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

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.xml.classgen.InvalidContentException
```

Implemented Interfaces of InvalidContentException

`java.io.Serializable`

Methods of InvalidContentException

InvalidContentException()

Description

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

Description of oracg

Provides a command-line interface to generate java classes corresponding to the DTD or XML

Syntax of oracg

```
public class oracg extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.classgen.oracg
```

Command-line options of oracg

Table 7–7 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

Description of SchemaClassGenerator

This class generates the classes corresponding to an XML Schema.

Syntax of SchemaClassGenerator

```
public class SchemaClassGenerator extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.classgen.SchemaClassGenerator
```

Methods of SchemaClassGenerator

Table 7–8 Summary of Methods of SchemaClassGenerator

Method	Description
SchemaClassGenerator() on page 7-25	Constructor.
generate() on page 7-26	Generates the Schema classes corresponding to top level elements, simpleType elements and complexType elements.
setGenerateComments() on page 7-26	Sets the switch to determine whether to generate java doc comments.
setJavaPackage() on page 7-27	Assigns a user-defined Java package name for each namespace.
setOutputDirectory() on page 7-27	Sets the output directory where the java source code for the Schema class are generated.

SchemaClassGenerator()

Description

Constructor. The options are described in the following table.

Syntax	Description
public SchemaClassGenerator();	Default empty constructor for Schema Class Generator.

Syntax	Description
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()

Description

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()

Description

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()

Description

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()

Description

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.

8

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:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

OracleXMLQuery Class

Description of OracleXMLQuery

The OracleXMLQuery class generates XML given an SQL query.

Syntax of Oracle XMLQuery

```
public class OracleXMLQuery extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.sql.query.OracleXMLQuery
```

Fields of OracleXMLQuery

Table 8–1 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.

Methods of OracleXMLQuery

Table 8–2 Summary of Methods of OracleXMLQuery

Method	Description
OracleXMLQuery() on page 8-4	Class constructor.
close() on page 8-5	Closes open resources created by the Oracle XML engine.
getNumRowsProcessed() on page 8-5	Returns the number of rows processed.
getXMLDOM() on page 8-6	Transforms data into an XML document.
getXMLMetaData() on page 8-6	Transforms object-relational data, specified in the constructor, into an XML document.
getXMLSAX() on page 8-7	Returns the DTD or XMLSchema for the XML document.
getXMLSchema() on page 8-7	Transforms object-relational data, specified in the constructor, into an XML document.
getXMLString() on page 8-8	Generates XMLSchema(s) corresponding to the specified query.
keepObjectOpen() on page 8-8	Transforms object-relational data, specified in the constructor, into an XML document.
removeXSLTParam() on page 8-9	Has the effect of turning on and off the persistency of objects from which XML data is retrieved.
setCollIdAttrName() on page 8-9	Removes the value of a top-level stylesheet parameter.
setDataHeader() on page 8-10	Sets the name of the id attribute of the collection element's separator tag.
setDateFormat() on page 8-10	Sets the XML data header.
setEncoding() on page 8-11	Sets the format of the generated dates in the XML doc.
setErrorTag() on page 8-11	Sets the encoding processing instruction in the XML doc.
setException() on page 8-11	Sets the tag to be used to enclose the XML error docs.
setMaxRows() on page 8-12	Allows the user to pass in an exception to be handled by the XSU.
setMetaHeader() on page 8-12	Sets the maximum number of rows to be converted to XML.
setRaiseException() on page 8-13	Sets the XML meta header.
setRaiseNoRowsException() on page 8-13	Instructs the XSU whether to throw the raised exceptions.

Table 8–2 Summary of Methods of OracleXMLQuery (Cont.)

Method	Description
setRowIdAttrName() on page 8-13	Instructs the XSU whether to throw an `OracleXMLNoRowsException` when the generated XML doc is empty.
setRowIdAttrValue() on page 8-14	Sets the name of the id attribute of the row enclosing tag.
setRowsetTag() on page 8-14	Specifies the scalar column whose value will be assigned to the id attribute of the row enclosing tag.
setRowTag() on page 8-14	Sets the tag to be used to enclose the XML dataset.
setSkipRows() on page 8-15	Sets the number of rows to skip.
setSQLToXMLNameEscaping() on page 8-15	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 8-16	Sets the stylesheet header.
setXSLT() on page 8-16	Registers an XSL transform to be applied to the generated XML.
setXSLTParam() on page 8-17	Sets the value of a top-level stylesheet parameter.
useLowerCaseTagNames() on page 8-17	Sets the tag names to lower case.
useNullAttributeIndicator() on page 8-17	Specifies if `NULL`ness is indicated by a special XML attribute or by omitting the entity from the XML document.
useTypeForCollElemTag() on page 8-18	Instructs the XSU to use the collection element's type name as the collection element’s tag name.
useUpperCaseTagNames() on page 8-18	Sets the tag names to upper case.

OracleXMLQuery()

Description

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.
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()

Description

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()

Description

Returns the number of rows processed.

Syntax

```
public long getNumRowsProcessed();
```

getXMLDOM()

Description

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.
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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Sets the tag to be used to enclose the XML error docs.

Syntax

```
public void setErrorTag( String tag);
```

Parameter	Description
tag	Tag name

setException()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Sets the tag to be used to enclose the XML dataset.

Syntax

```
public void setRowsetTag( String tag);
```

Parameter	Description
tag	Tag name

setRowTag()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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.
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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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

Description of OracleXMLSave

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 of OracleXMLSave

```
public class OracleXMLSave extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.sql.dml.OracleXMLSave
```

Fields of OracleXMLSave

Table 8–3 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.

Methods of OracleXMLSave

Table 8–4 Summary of Methods of OracleXMLSave

Method	Description
OracleXMLSave() on page 8-20	The public constructor for the Save class.
close() on page 8-21	It closes / deallocates all the context associated with this object.
deleteXML() on page 8-21	Deletes the rows in the table based on the XML document.

Table 8–4 Summary of Methods of OracleXMLSave (Cont.)

Method	Description
getURL() on page 8-22	Return a URL object given a file name or a URL.
insertXML() on page 8-23	Inserts an XML document into the specified table.
removeXSLTParam() on page 8-24	Removes the value of a top-level stylesheet parameter.
setBatchSize() on page 8-24	Changes the batch size used during DML operations.
setCommitBatch() on page 8-24	Sets the commit batch size.
setDateFormat() on page 8-25	Describes to the XSU the format of the dates in the XML document.
setIgnoreCase() on page 8-25	Instructs the XSU to perform a case-insensitive match of XML elements to database columns or attributes.
setKeyColumnList() on page 8-26	Sets the list of columns to be used for identifying a particular row in the database table during update or delete.
setPreserveWhitespace() on page 8-26	Instructs the XSU whether to preserve whitespaces.
setRowTag() on page 8-27	Names the tag used in the XML doc. to enclose the XML elements corresponding to each row value.
setSQLToXMLNameEscaping() on page 8-27	This turns on or off escaping of XML tags when an SQL object name would not make a valid XML identifier.
setUpdateColumnList() on page 8-27	Set the column values to be updated.
setXSLT() on page 8-28	Registers a XSL transform to be applied to generated XML.
setXSLTParam() on page 8-29	Sets the value of a top-level stylesheet parameter.
updateXML() on page 8-29	Updates the table given the XML document.

OracleXMLSave()

Description

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()

Description

Closes / deallocates all the context associated with this object.

Syntax

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

deleteXML()

Description

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.

Syntax	Description
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.
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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Instructs the XSU whether to preserve whitespaces.

Syntax

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

Parameter	Description
flag	Should the whitespaces be preserved?

setRowTag()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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 which are used to uniquely identify a row to update in the given table. 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, subsequently updating all the columns in the table for which there is an equivalent element present in the XML document. Each ROW element present in the input document 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

Description of OracleXMLSQLException

Class for managing all exceptions thrown by the XSU.

Syntax of OracleXMLSQLException

```
public class OracleXMLSQLException extends java.lang.RuntimeException

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--oracle.xml.sql.OracleXMLSQLException
```

Direct Subclasses of OracleXMLSQLException

- `OracleXMLSQLNoRowsException Class`

Implemented Interfaces of OracleXMLSQLException

- `java.io.Serializable`

Methods of OracleXMLSQLException

Table 8–5 Summary of Methods of OracleXMLSQLException

Method	Description
OracleXMLSQLException() on page 8-32	Creates a new OracleXMLSQLException.
getErrorCode() on page 8-33	Returns the SQL error code thrown.
getParentException() on page 8-33	Returns the original exception if there was one; otherwise, returns `NULL`.
getXMLErrorString() on page 8-33	Prints XML error string with given error tag name.
getXMLSQLErrorString() on page 8-34	Prints the SQL parameters in the error message.

Table 8–5 Summary of Methods of OracleXMLSQLException (Cont.)

Method	Description
setErrorTag() on page 8-34	Sets error tag used to generate XML error reports.

OracleXMLSQLException()

Description

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.
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.

Syntax	Description

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()

Description

Returns the SQL error code thrown.

Syntax

```
public int getErrorCode();
```

getParentException()

Description

Returns the original exception, if there was one; otherwise, returns `NULL`.

Syntax

```
public java.lang.Exception getParentException();
```

getXMLErrorString()

Description

Prints the XML error string with the given error tag name.

Syntax

```
public String getXMLErrorString();
```

getXMLSQLErrorString()

Description

Prints the SQL parameters as well in the error message.

Syntax

```
public String getXMLSQLErrorString();
```

setErrorTag()

Description

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

Description of OracleXMLSQLNoRowsException

The exception that can be thrown when no rows are found.

Syntax of OracleXMLSQLNoRowsException

```
public class OracleXMLSQLNoRowsException extends OracleXMLSQLException

java.lang.Object
 |
 +--java.lang.Throwable
    |
    +--java.lang.Exception
       |
       +--java.lang.RuntimeException
          |
          +--OracleXMLSQLException
             |
             +--oracle.xml.sql.OracleXMLSQLNoRowsException
```

Implemented Interfaces of OracleXMLSQLNoRowsException

`java.io.Serializable`

Methods of OracleXMLSQLNoRowsException

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.

9 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:

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

oracle.xml.xsql Package

Description of oracle.xml.xsql

The Oracle XSQL Pages Publishing Framework engine. Table 9–1 summarizes the classes and interfaces of this package.

Table 9–1 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 Interfacet	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

Description of XSQLActionHandler

Interface that must be 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
- Invoking the XSQL Action Handler's handleAction() method

NOTE: conn parameter can be null if no connection specified for the XSQL page being processed.

Syntax of XSQLActionHandler

```
public interface XSQLActionHandler
```

Implementing Classes of XSQLActionHandler

`XSQLActionHandlerImpl Class`

Methods of XSQLActionHandler

Table 9–2 Summary of Methods of XSDLAction Handler

Method	Description
handleAction() on page 9-3	Handle the action, typically by executing some code and appending new child DOM nodes to the rootNode.
init() on page 9-4	Initializes the Action Handler.

handleAction()

Description

Handle the action, typically by executing some code and appending new child DOM nodes to the rootNode.

The XSQL Page Processor replaces the action element in the XSQL Page being processed with the document fragment of nodes that your handleAction method appends to the rootNode.

Syntax

```
public void handleAction( oracle.xml.xsql.Node rootNode);
```

Parameter	Description
rootNode	Root node of generated document fragment.

init()

Description

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

Description of XSQLActionHandlerImpl

Base Implementation of XSQLActionHandler that can be extended to create 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 of XSQLActionHandlerImpl

```
public abstract class XSQLActionHandlerImpl extends java.lang.Object implements
XSQLActionHandler Interface

java.lang.Object
  |
  +--oracle.xml.xsql.XSQLActionHandlerImpl
```

Implemented Interfaces of XSQLActionHandlerImpl

`XSQLActionHandler Interface`

Methods of XSQLActionHandlerImpl

Table 9–3 Summary of Methods of XSQLActionHandlerImp

Method	Description
XSQLActionHandlerImpl() on page 9-5	Class constructor.
init() on page 9-6	Initializes the action handler.

XSQLActionHandlerImpl()

Description

Class constructor.

Syntax

```
public  XSQLActionHandlerImpl();
```

init()

Description

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

Description of XSQLPageRequest

Interface representing a request for an XSQL Page.

Syntax of XSQLPageRequest

`public interface XSQLPageRequest`

Implementing Classes of XSQLPageRequest

`XSQLPageRequestImpl Class`

Methods of XSQLPageRequest

Table 9–4 Summary of Methods of XSQLPageRequest()

Method	Description
createNestedRequest() on page 9-9	Returns an instance of a nested Request
getConnectionName() on page 9-9	Returns the name of the connection being used for this request May be null if no connection set/in-use.
getErrorWriter() on page 9-9	Returns a PrintWriter to print out errors processing this request
getJDBCConnection() on page 9-10	Gets the JDBC connection being used for this request (can be null)
getPageEncoding() on page 9-10	Returns encoding of source XSQL Page associated with this request
getParameter() on page 9-10	Returns the value of the requested parameter
getPostedDocument() on page 9-10	Returns the content of Posted XML for this request as an XML Document
getRequestParamsAsXMLDocument() on page 9-11	Returns the content of a Request parameters as an XML Document
getRequestType() on page 9-11	Returns a string identifying the type of page request being made.
getSourceDocumentURI() on page 9-11	Returns a String representation of the requested document's URI
getStylesheetParameter() on page 9-11	Gets a stylesheet parameter by name

Table 9–4 Summary of Methods of XSQLPageRequest() (Cont.)

Method	Description
getStylesheetParameters() on page 9-12	Gets an enumeration of stylesheet parameter names
getStylesheetURI() on page 9-12	Returns the URI of the stylesheet to be used to process the result.
getUserAgent() on page 9-12	Returns a String identifier of the requesting program
getWriter() on page 9-12	Returns a PrintWriter used for writing out the results of a page request
getXSQLConnection() on page 9-13	Gets the XSQLConnection Object being used for this request Might be null.
isIncludedRequest() on page 9-13	Returns true if this request is being included in another.
isOracleDriver() on page 9-13	Returns true if the current connection uses the Oracle JDBC Driver
printedErrorHeader() on page 9-13	Returns the state of whether an Error Header has been printed
requestProcessed() on page 9-13	Allows Page Request to Perform end-of-request processing
setConnectionName() on page 9-14	Sets the connection name to use for this request
setContentType() on page 9-14	Sets the content type of the resulting page
setIncludingRequest() on page 9-14	Sets the Including Page Request object for this request.
setPageEncoding() on page 9-15	Sets encoding of source XSQL page associated with this request.
setPageParam() on page 9-15	Sets a dynamic page parameter value.
setPostedDocument() on page 9-15	Allows programmatic setting of the Posted Document
setPrintedErrorHeader() on page 9-16	Sets whether an Error Header has been printed
setStylesheetParameter() on page 9-16	Sets the value of a parameter to be passed to the associated stylesheet
setStylesheetURI() on page 9-17	Sets the URI of the stylesheet to be used to process the result.
translateURL() on page 9-17	Returns a string representing an absolute URL resolved relative to the base URI for this request.
useConnectionPooling() on page 9-17	Returns true if connection pooling is desired for this request

Table 9–4 Summary of Methods of XSQLPageRequest() (Cont.)

Method	Description
useHTMLErrors() on page 9-18	Returns true if HTML-formatted error messages are desired for this request.
setRequestObject() on page 9-18	Sets a request-scope object.
getRequestObject() on page 9-18	Gets a request-scope object.

createNestedRequest()

Description

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()

Description

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()

Description

Returns a PrintWriter to print out errors processing this request.

Syntax

```
public java.io.PrintWriter getErrorWriter();
```

getJDBCConnection()

Description

Gets the JDBC connection being used for this request (can be null).

Syntax

```
public java.sql.Connection getJDBCConnection();
```

getPageEncoding()

Description

Returns encoding of source XSQL Page associated with this request.

Syntax

```
public java.lang.String getPageEncoding();
```

getParameter()

Description

Returns the value of the requested parameter.

Syntax

```
public String getParameter( String name);
```

Parameter	Description
name	The name of the parameter.

getPostedDocument()

Description

Returns the content of Posted XML for this request as an XML Document.

Syntax

```
public oracle.xml.xsql.Document getPostedDocument();
```

getRequestParamsAsXMLDocument()

Description

Returns the content of a Request parameters as an XML Document.

Syntax

```
public oracle.xml.xsql.Document getRequestParamsAsXMLDocument();
```

getRequestType()

Description

Returns a string identifying the type of page request being made.

Syntax

```
public String getRequestType();
```

getSourceDocumentURI()

Description

Returns a String representation of the requested document's URI.

Syntax

```
public String getSourceDocumentURI();
```

getStylesheetParameter()

Description

Gets a stylesheet parameter by name.

Syntax

```
public String getStylesheetParameter( String name);
```

Parameter	Description
name	The stylesheet parameter name.

getStylesheetParameters()

Description

Gets an enumeration of stylesheet parameter names.

Syntax

```
public java.util.Enumeration getStylesheetParameters();
```

getStylesheetURI()

Description

Returns the URI of the stylesheet to be used to process the result.

Syntax

```
public java.lang.String getStylesheetURI();
```

getUserAgent()

Description

Returns a String identifier of the requesting program.

Syntax

```
public java.lang.String getUserAgent();
```

getWriter()

Description

Returns a PrintWriter used for writing out the results of a page request.

Syntax

```
public java.io.PrintWriter getWriter();
```

getXSQLConnection()

Description

Gets the XSQLConnection Object being used for this request Might be null.

Syntax

```
public oracle.xml.xsql.XSQLConnection getXSQLConnection();
```

isIncludedRequest()

Description

Returns true if this request is being included in another.

Syntax

```
public boolean isIncludedRequest();
```

isOracleDriver()

Description

Returns true if the current connection uses the Oracle JDBC Driver.

Syntax

```
public boolean isOracleDriver();
```

printedErrorHeader()

Description

Returns the state of whether an Error Header has been printed.

Syntax

```
public boolean printedErrorHeader();
```

requestProcessed()

Description

Allows Page Request to Perform end-of-request processing.

Syntax

```
public void requestProcessed();
```

setConnectionName()

Description

Sets the connection name to use for this request.

Syntax

```
public void setConnectionName( String connName);
```

Parameter	Description
connName	The name of the connection.

setContentType()

Description

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()

Description

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()

Description
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()

Description
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()

Description
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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Returns true if connection pooling is desired for this request.

Syntax

```
public boolean useConnectionPooling();
```

useHTMLErrors()

Description

Returns true if HTML-formatted error messages are desired for this request.

Syntax

```
public boolean useHTMLErrors();
```

setRequestObject()

Description

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()

Description

Gets a request-scope object.

Syntax

```
Object getRequestObject( String name);
```

Parameter	Description
name	Name of the request-scope object to retrieve.

XSQLParserHelper Class

Description of XSQLParserHelper

Common XML Parsing Routines.

Syntax of XSQLParserHelper

```
public final class XSQLParserHelper extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.xsql.XSQLParserHelper
```

Methods of XSQLParserHelper

Table 9–5 Summary of Methods of XSQLParserHelper

Method	Description
XSQLParserHelper() on page 9-19	Class constructor.
newDocument() on page 9-19	Returns a new, empty XML document.
parse() on page 9-20	Parses an XML document from a variety of sources.
parseFromString() on page 9-21	Parses an XML document from a string.
print() on page 9-21	Prints an XML document.

XSQLParserHelper()

Description

Class constructor.

Syntax

```
public  XSQLParserHelper();
```

newDocument()

Description

Returns a new, empty XML document.

Syntax

```
public static oracle.xml.xsql.Document newDocument();
```

parse()

Description

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()

Description

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()

Description

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

Description of XSQLRequest

Programmatically process a request for an XSQL Page.

Syntax of XSQLRequest

```
public class XSQLRequest extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.xsql.XSQLRequest
```

Methods of XSQLRequest

Table 9–6 Summary of Methods of XSQLRequest

Method	Description
XSQLRequest() on page 9-22	Class constructor. Create a Request for an XSQL Page
process() on page 9-23	Process the request, writing output/errors.
processToXML() on page 9-23	Process the request, writing output/errors.
setPostedDocument() on page 9-24	Programmatically set an XML Document to be treated the same as if it were posted as part of the request.

XSQLRequest()

Description

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()

Description

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.
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()

Description

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()

Description

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

Description of XSQLRequestObjectListener

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.

Methods of XSQLRequestObjectListener

pageProcessingCompleted()

Description

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 of XSQLServletPageRequest

```
public final class XSQLServletPageRequest extends XSQLPageRequestImpl

java.lang.Object
  |
  +--XSQLPageRequestImpl
        |
        +--oracle.xml.xsql.XSQLServletPageRequest
```

Implemented Interfaces of XSQLServletPageRequest

`XSQLPageRequest Interface`

Methods of XSQLServletPageRequest

Table 9–7 Summary of Methods of XSQLServletPageRequest

Method	Description
XSQLServletPageRequest() on page 9-27	Creates a Servlet-specific XSQL Page request.
createNestedRequest() on page 9-27	Returns an instance of a nested Request.
getCookie() on page 9-28	Gets a vale of an HTTP Request cookie.
getHttpServletRequest() on page 9-28	Get the HttpServletRequest that initiated this XSQL Page Request.
getHttpServletResponse() on page 9-28	Get the HttpServletResponse that is associated with this XSQL Page Request.
getParameter() on page 9-28	Use HTTP Parameters as the source of parameters instead.
getPostedDocument() on page 9-29	Retrieves a posted XML document for this request.
getRequestParamsAsXMLDocument() on page 9-29	Retrieves request parameters as an XML document.
getRequestType() on page 9-29	Returns the type of request.
getUserAgent() on page 9-29	Returns the user agent string passed in from the browser.
setContentType() on page 9-30	Sets the content type of the resulting page.

Table 9–7 Summary of Methods of XSQLServletPageRequest (Cont.)

Method	Description
setPageEncoding() on page 9-30	Sets the page encoding.
translateURL() on page 9-30	Returns a string representing an absolute URL resolved relative to the base URI for this request.
useHTMLErrors() on page 9-31	Sets whether HTML-formatted errors should be used.

XSQLServletPageRequest()

Description

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()

Description

Returns an instance of a nested Request.

Syntax

```
public XSQLPageRequest createNestedRequest(
                   java.net.URL pageurl,
                   java.util.Dictionary params)
```

Parameter	Description
pageurl	Page URL.

Parameter	Description
prams	Parameters of the request.

getCookie()

Description
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()

Description
Get the HttpServletRequest that initiated this XSQL Page Request.

Syntax
```
public oracle.xml.xsql.HttpServletRequest getHttpServletRequest();
```

getHttpServletResponse()

Description
Get the HttpServletResponse that is associated with this XSQL Page Request.

Syntax
```
public oracle.xml.xsql.HttpServletResponse getHttpServletResponse();
```

getParameter()

Description
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()

Description

Retrieves a posted XML document for this request. Returns `NULL` if there is none.

Syntax

```
public oracle.xml.xsql.Document getPostedDocument();
```

getRequestParamsAsXMLDocument()

Description

Retrieves request parameters as an XML document.

Syntax

```
public oracle.xml.xsql.Document getRequestParamsAsXMLDocument();
```

getRequestType()

Description

Returns the type of request ("Servlet", "Programmatic", "Command Line", or "Custorm").

Syntax

```
public java.lang.String getRequestType();
```

getUserAgent()

Description

Returns the user agent string passed in from the browser.

Syntax

```
public java.lang.String getUserAgent();
```

setContentType()

Description

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()

Description

Sets the page encoding.

Syntax

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

Parameter	Description
enc	The page encoding.

translateURL()

Description

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()

Description

Sets whether HTML-formatted errors should be used.

Syntax

```
public boolean useHTMLErrors();
```

XSQLStylesheetProcessor Class

Description of XSQLStylesheetProcessor

XSLT Stylesheet Processing Engine

Syntax of XSQLStylesheetProcessor

```
public final class XSQLStylesheetProcessor extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.xsql.XSQLStylesheetProcessor
```

Methods of XSQLStylesheetProcessor

Table 9–8 Summary of Methods of XSQLStylesheetProcessor

Method	Description
XSQLStylesheetProcessor() on page 9-32	Processes CSLT stylesheet transformations.
processToDocument() on page 9-33	Transforms an XML document by an XSLT stylesheet and returns the result as an XML document.
processToWriter() on page 9-33	Transforms an CML Document by an XSLT stylesheet and writes the result to a PrintWriter.
getServletContext() on page 9-34	Gets the Http Servlet Context.

XSQLStylesheetProcessor()

Description

Processes CSLT stylesheet transformations.

Syntax

```
public  XSQLStylesheetProcessor();
```

processToDocument()

Description

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.
env	The XSQLPageRequest context.

processToWriter()

Description

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()

Description

Gets the Http Servlet Context.

Syntax

```
javax.servlet.ServletContext getServletContext();
```

XSQLConnectionManager Interface

Description of XSQLConnectionManager

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 of XSQLConnectionManager

```
public interface XSQLConnectionManager;
```

Method of XSQLConnectionManager

Table 9–9 Summary of Methods of XSQLConnectionManager

Method	Description
getConnection() on page 9-35	Gets a connection from the connection manager.
releaseConnection() on page 9-36	Releases the connection.

getConnection()

Description

Gets a connection from the connection manager.

Syntax

```
XSQLConnection getConnection( String connName,
                              XSQLPageRequest env);
```

Parameter	Description
connName	The connection name.

Parameter	Description
env	The XSQLPageRequest context.

releaseConnection()

Description

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

Description of XSQLConnectionManagerFactory

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 of XSQLConnectionManagerFactory

```
public interface XSQLConnectionManagerFactory
```

Methods of XSQLConnectionManagerFactory

create()

Description

Returns an instance of an XSQLConnectionManager. Implementation can decide whether it is a new XSQLConnectionManager or a shared singleton.

Syntax

```
XSQLConnectionManager create();
```

XSQLDocumentSerializer Interface

Description of XSQLDocumentSerializer

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

NOTE: 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.

Syntax of XSQLDocumentSerializer

```
public interface XSQLDocumentSerializer
```

Method of XSQLDocumentSerialize

serialize()

Description

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.

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:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

oracle.xml.async Package

Description

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 10-14	This class provides information about the DOMBuilder Bean.
DOMBuilderErrorEvent Class on page 10-16	This class defines the error event which is sent when parse exception occurs.
DOMBuilderErrorListener Interface on page 18	This interface must be implemented in order to receive notifications when error is found during parsing.
DOMBuilderEvent Class on page 10-19	The event object that DOMBuilder uses to notify all registered listeners about parse events.
DOMBuilderListener Interface on page 10-21	This interface must be implemented in order to receive notifications about events during the asyncronous parsing.
ResourceManager Class on page 10-23	This class implements a simple semaphore that maintains access to a fixed number of logical resources.
XSLTransformer Class on page 25	Applies XSL transformation in a background thread.
XSLTransformerBeanInfo Class on page 10-31	This class provides information about the XSLTransformer Bean.
XSLTransformerErrorEvent Class on page 10-33	The error event object that XSLTransformer uses to notify all registered listeners about transformation error events.

DOMBuilder Class

Description of DOMBuilder

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 of DOMBuilder

```
public class DOMBuilder extends java.lang.Object implements
java.io.Serializable, oracle.xml.async.DOMBuilderConstants, java.lang.Runnable

java.lang.Object
  |
  +--oracle.xml.async.DOMBuilder
```

Implemented Interfaces

- `oracle.xml.async.DOMBuilderConstants`
- `java.lang.Runnable`
- `java.io.Serializable`

Fields of DOMBuilder

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

Table 10–2 Fields of DOMBuilder (Cont.)

Field	Syntax	Description
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

Methods of DOMBuilder

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 10-6	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 10-7	Returns the release version of Oracle XML Parser.
getResult() on page 10-7	Gets the document being parsed.
getValidationMode() on page 10-7	Returns the validation mode.
parse() on page 10-7	Parses the XML from given input.
parseDTD() on page 10-8	Parses the XML External DTD.
removeDOMBuilderErrorListener() on page 10-9	Removes the DOMBuilderErrorListener.
removeDOMBuilderListener() on page 10-10	Removes the DOMBuilderListener.
run() on page 10-10	Runs in a thread.
setBaseURL() on page 10-10	Sets the base URL for lading external entities and DTD.
setDebugMode() on page 10-11	Sets a flag to run on debug information in the document.

Table 10–3 Summary of Methods of DOMBuilder (Cont.)

Method	Description
setDoctype() on page 10-11	Sets the DTD.
setErrorStream() on page 10-12	Creates an output stream for errors and warnings.
setNodeFactory() on page 10-12	Sets the node factory.
setPreserveWhitespace() on page 10-13	Sets the white space preservation mode.
setValidationMode() on page 10-13	Sets the validation mode.
showWarnings() on page 10-13	Determines whether to print warnings.

DOMBuilder()

Description

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()

Description

Adds DOMBuilderErrorListener.

Syntax

```
public void addDOMBuilderErrorListener( DOMBuilderErrorListener p0);
```

Parameter	Description
p0	The `DOMBuilderErrorListener` to add.

addDOMBuilderListener()

Description
Adds DOMBuilderListener

Syntax
```
public void addDOMBuilderListener(DOMBuilderListener p0);
```

Parameter	Description
p0	The DOMBuilderListener to add.

getDoctype()

Description
Gets the DTD.

Syntax
```
public synchronized oracle.xml.async.DTD getDoctype();
```

getDocument()

Description
Gets the document for parsing.

Syntax
```
public synchronized oracle.xml.async.XMLDocument getDocument();
```

getId()

Description
Returns the parser object id (DOMBuilder).

Syntax

```
public int getId();
```

getReleaseVersion()

Description

Returns the release version of the Oracle XML Parser, as a String.

Syntax

```
public synchronized java.lang.String getReleaseVersion();
```

getResult()

Description

Gets the document being parsed.

Syntax

```
public synchronized org.w3c.dom.Document getResult();
```

getValidationMode()

Description

Returns the validation mode; TRUE if XML parser is validating, FALSE otherwise.

Syntax

```
public synchronized boolean getValidationMode()
```

parse()

Description

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.

The options are described in the following table.

Syntax	Description
public final synchronized void parse (InputSource in);	Parses the XML from an InputSource
public final synchronized void parse (j 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()

Description

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.

The options are described in the following table.

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()

Description

Removes DOMBuilderErrorListener.

Syntax

```
public synchronized void removeDOMBuilderErrorListener(
                  DOMBuilderErrorListener p0);
```

Parameter	Description
po	The DOMBuilderErrorListener to remove.

removeDOMBuilderListener()

Description

Removes DOMBuilderListener.

Syntax

```
public synchronized void removeDOMBuilderListener(
                  DOMBuilderListener p0);
```

Parameter	Description
p1	The DOMBuilderListener to remove.

run()

Description

This method runs in a thread. It is specified in java.lang.Runnable.run() in interface java.lang.Runnable.

Syntax

```
public void run();
```

setBaseURL()

Description

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()

Description

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 information is stored, FALSE otherwise.

setDoctype()

Description

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()

Description

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()

Description

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 the following exception:

XMLParseException	If an invalid factory is set.

Syntax

```
public synchronized void setNodeFactory(
                  oracle.xml.async.NodeFactory factory);
```

Parameter	Description
factory	The NodeFactory to set.

setPreserveWhitespace()

Description

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()

Description

Sets the validation mode.

Syntax

```
public synchronized void setValidationMode(boolean yes);
```

Parameter	Description
yes	Determines whether the XML parser should be validating; TRUE for validation, FALSE otherwise.

showWarnings()

Description

Determines whether to print warnings.

Syntax

```
public synchronized void showWarnings(boolean yes);
```

Parameter	Description
yes	Switch; TRUE to print warnings, FALSE otherwise.

DOMBuilderBeanInfo Class

Description of DOMBuilderBeanInfo

This class provides information about the DOMBuilder Bean.

Syntax of DOMBuilderBeanInfo

```
public class DOMBuilderBeanInfo extends java.beans.SimpleBeanInfo

java.lang.Object
  |
  +--java.beans.SimpleBeanInfo
        |
        +--oracle.xml.async.DOMBuilderBeanInfo
```

Implemented Interfaces of DOMBuilderBeanInfo

- `java.beans.BeanInfo`

Methods of DOMBuilderBeanInfo

Table 10–4 Summary of Methods of DOMBuilderBeanInfo

Method	Description
DOMBuilderBeanInfo() on page 10-14	The default constructor.
getIcon() on page 10-15	Gets an image object that can be used to represent the DOMBuilder bean in toolbars, toolboxes, and so on.
getPropertyDescriptors() on page 10-15	Retrieves the array of DOMBuilder bean's editable PropertyDescriptors.

DOMBuilderBeanInfo()

Description

The default constructor.

Syntax

```
public  DOMBuilderBeanInfo();
```

getIcon()

Description

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. Overrides `getIcon()` method in `java.beans.SimpleBeanInfo` Class.

Syntax

```
public java.awt.Image getIcon( int iconKind);
```

Parameter	Description
iconKind	The kind of icon requested.

getPropertyDescriptors()

Description

Retrieves the array of DOMBuilder bean's editable PropertyDescriptiors. Overrides `getPropertyDescriptors()` in `java.beans.SimpleBeanInfo` Class.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

DOMBuilderErrorEvent Class

Description of DOMBuilderErrorEvent

This class defines the error event which is sent when parse exception occurs.

Syntax of DOMBuilderErrorEvent

```
public class DOMBuilderErrorEvent extends java.util.EventObject

java.lang.Object
  |
  +--java.util.EventObject
        |
        +--oracle.xml.async.DOMBuilderErrorEvent
```

Implemented Interfaces of DOMBuilderErrorEvent

- `java.io.Serializable`

Fields of DOMBuilderErrorEvent

Table 10–5 Fields of DOMBuilderErrorEvent

Field	Syntax	Description
e	protected java.lang.Exception	The exception being raised.

Methods of DOMBuilderErrorEvent

Table 10–6 Summary of Methods of DOMBuilderErrorEvent

Method	Description
DOMBuilderErrorEvent() on page 10-17	Constructor for DOMBuilderErrorEvent.
getException() on page 10-17	Retrieves the exception being raised.
getMessage() on page 10-17	Returns the error message generated by the parser.

DOMBuilderErrorEvent()

Description

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()

Description

Retrieves the exception being raised.

Syntax

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

getMessage()

Description

Returns the error message generated by the parser, as a String.

Syntax

```
public java.lang.String getMessage();
```

DOMBuilderErrorListener Interface

Description of DOMBuilderErrorListener

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 of DOMBuilderErrorListener

```
public interface DOMBuilderErrorListener extends java.util.EventListener
```

Methods of DOMBuilderErrorListener

domBuilderErrorCalled()

Description

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

Description of DOMBuilderEvent

The event object that DOMBuilder uses to notify all registered listeners about parse events.

Syntax of DOMBuilderEvent

```
public class DOMBuilderEvent extends java.util.EventObject

java.lang.Object
  |
  +--java.util.EventObject
        |
        +--oracle.xml.async.DOMBuilderEvent
```

Implemented Interfaces of DOMBuilderEvent

- `java.io.Serializable`

Fields of DOMBuilderEvent

Table 10–7 Fields of DOMBuilderEvent

Field	Syntax	Description
id	protected int id	ID of the source DOMBuilder object

Methods of DOMBuilderEvent

Table 10–8 Summary of Methods of DOMBuilderEvent

Method	Description
DOMBuilderEvent() on page 10-19	Creates a new DOMBuilderEvent.
getID() on page 10-20	Returns unique id of the source DOMBuilder for the event.

DOMBuilderEvent()

Description

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()

Description

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

Description of DOMBuilderListener

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 of DOMBuilderListener

```
public interface DOMBuilderListener extends java.util.EventListener
```

Methods of DOMBuilderListener

Table 10–9 Summary of Methods of DOMBuilderListener

Method	Description
domBuilderError() on page 10-18	This method is called when parse error occur.
domBuilderOver() on page 10-22	This method is called when the parse is complete.
domBuilderStarted() on page 10-22	This method is called when parse starts

domBuilderError()

Description

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()

Description

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()

Description

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

Description of ResourceManager

Implements a semaphore and maintains access to a fixed number of logical resources.

Syntax of ResourceManager

```
public class ResourceManager extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.async.ResourceManager
```

Methods of ResourceManager

Table 10–10 Summary of Methods of ResourceManager

Method	Description
ResourceManager() on page 10-23	The ResourceManager constructor.
activeFound() on page 10-24	Checks if any of the logical resources being managed are in active use.
getResource() on page 10-24	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 10-24	Releases a single resource, increasing the number of resources available.
sleep() on page 10-24	Allows use of Thread.sleep() without try/catch.

ResourceManager()

Description

The ResourceManager constructor.

Syntax

```
public  ResourceManager(int i);
```

Parameter	Description
i	The number of resources to manage.

activeFound()

Description

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()

Description

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()

Description

Releases a resource. When this method is called, the number of resources available is increased by one.

Syntax

```
public void releaseResource();
```

sleep()

Description

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

Description of XSLTransformer

Applies XSL transformation in a background thread.

Syntax of XSLTransformer

```
public class XSLTransformer extends java.lang.Object implements
    java.io.Serializable, oracle.xml.async.XSLTransformerConstants,
    java.lang.Runnable

java.lang.Object
  |
  +--oracle.xml.async.XSLTransformer
```

Fields of XSLTransformer

Table 10–11 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.

Methods of XSLTransformer

Table 10–12 Summary of Methods of XSLTransformer

Method	Description
XSLTransformer() on page 10-26	XSLTransformer class constructor.
addXSLTransformerErrorListener() on page 10-26	Adds and XSLTransformerErrorListener.
addXSLTransformerListener() on page 10-27	Adds and XSLTransformerListener.
getId() on page 10-27	Returns a unique XSLTransformer id.
getResult() on page 10-27	Returns the document fragment of the XSL transformation for the resulting document.

Table 10–12 Summary of Methods of XSLTransformer (Cont.)

Method	Description
processXSL() on page 10-28	Initiates XSL Transformation in the background. The control is returned immediately.
removeDOMTransformerErrorListener() on page 10-29	Removes an XSLTransformerErrorListener.
removeXSLTransformerListener() on page 10-29	Removes an XSLTransformerListener.
run() on page 10-30	Starts a separate thread to perform the XSLTransformation.
setErrorStream() on page 10-30	Sets the error stream to be used by XML processor.
showWarnings() on page 10-30	Sets the showWarnings flag used by the XSL processor.

XSLTransformer()

Description

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	A unique integer that can be used to identify the XSLTransformer instance during event processing

addXSLTransformerErrorListener()

Description

Adds an XSLTransformerErrortListener.

Syntax

```
public void addXSLTransformerErrorListener(
                  XSLTransformerErrorListerner p0);
```

Parameter	Description
p0	XSLTransformerErrorListener to be added.

addXSLTransformerListener()

Description

Adds a XSLTransformerListener.

Syntax

```
public void addXSLTransformerListener( XSLTransformerListerner p0);
```

Parameter	Description
p0	XSLTransformerListener to be added.

getId()

Description

Returns the unique XSLTransformer id.

Syntax

```
public int getId();
```

getResult()

Description

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()

Description

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.
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()

Description
Removes an XSLTransformerErrorListener.

Syntax
```
public synchronized void removeDOMTransformerErrorListener(
                  XSLTransformerErrorListener p0);
```

Parameter	Description
p0	The XSLTransformerErrorListener to be removed.

removeXSLTransformerListener()

Description
Removes a XSLTransformerListener.

Syntax
```
public synchronized void removeXSLTransformerListener(
                  XSLTransformerListener p0);
```

Parameter	Description
p0	The XSLTransformerListener to be removed.

run()

Description

Starts a separate thread to perform the XSLTransformation. Specified by java.lang.Runnable.run() in java.lang.Runnable Interface.

Syntax

```
public void run();
```

setErrorStream()

Description

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()

Description

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

Description of XSLTransformerBeanInfo

This class provides information about the XSLTransformer Bean.

Syntax of XSLTransformerBeanInfo

```
public class XSLTransformerBeanInfo extends java.beans.SimpleBeanInfo

java.lang.Object
  |
  +--java.beans.SimpleBeanInfo
        |
        +--oracle.xml.async.XSLTransformerBeanInfo
```

Implemented Interfaces of XSLTransformerBeanInfo

`java.beans.BeanInfo`

Methods of XSLTransformerBeanInfo

Table 10–13   Summary of Methods of XSLTransformerBeanInfo

Method	Description
XSLTransformerBeanInfo() on page 10-31	The default constructor.
getIcon() on page 10-32	Retrieves an image object representing the requested icon type for XSLTransformer bean in toolbars, toolboxes, and so on.
getPropertyDescriptors() on page 10-32	Retrieves the array of XSLTransformer bean's editable PropertyDescriptiors.

XSLTransformerBeanInfo()

Description

The default Constructor.

Syntax

```
public  XSLTransformerBeanInfo();
```

getIcon()

Description

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()

Description

Retrieves the array of XSLTransformer bean's editable PropertyDescriptiors. Overrides `getPropertyDescriptors()` in `java.beans.SimpleBeanInfo` class.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

XSLTransformerErrorEvent Class

Description of XSLTransformerErrorEvent

The error event object that XSLTransformer uses to notify all registered listeners about transformation error events.

Syntax of XSLTransformerErrorEvent

```
public class XSLTransformerErrorEvent extends java.util.EventObject

java.lang.Object
  |
  +--java.util.EventObject
        |
        +--oracle.xml.async.XSLTransformerErrorEvent
```

Implemented Interfaces of XSLTransformerErrorEvent

```
java.io.Serializable
```

Fields of XSLTransformerErrorEvent

Table 10–14 Fields of XSLTransformerErrorEvent

Field	Syntax	Description
e	protected java.lang.Exception e	The exception being raised.

Methods of XSLTransformerErrorEvent

Table 10–15 Summary of Methods of XSLTransformerErrorEvent

Method	Description
XSLTransformerErrorEvent() on page 10-34	Constructor for XSLTransformerErrorEvent.
getException() on page 10-34	Returns the transformation exception that XSLTransformer encountered an object unique id.
getMessage() on page 10-34	Returns the error message that describes the error encountered by XSLTransformer.

XSLTransformerErrorEvent()

Description

Constructor for XSLTransformerErrorEvent.

Syntax

```
public  XSLTransformerErrorEvent( Object p0,
                                   Exception e);
```

Parameter	Description
p0	The Object that created this event.
e	The exception raised.

getException()

Description

Returns the transformation exception that XSLTransformer encountered an object unique id.

Syntax

```
public Exception getException();
```

getMessage()

Description

Returns the error message that describes the error encountered by XSLTransformer.

Syntax

```
public String getMessage();
```

XSLTransformerErrorListener Interface

Description of XSLTransformerErrorListenerInterface

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 of XSLTransformerErrorListenerInterface

```
public interface XSLTransformerErrorListener extends
     java.util.EventListener
```

Methods of XSLTransformerErrorListenerInterface

xslTransformerErrorCalled()

Description

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

Description of XSLTransformerEvent

This class represents the event object used by XSLTransformer to notify XSL transformation events to all its registered listeners.

Syntax of XSLTransformerEvent

```
public class XSLTransformerEvent extends java.util.EventObject

java.lang.Object
  |
  +--java.util.EventObject
        |
        +--oracle.xml.async.XSLTransformerEvent
```

Implemented Interfaces of XSLTransformerEvent

`java.io.Serializable`

Fields of XSLTransformerEvent

Table 10–16 Fields of XSLTransformerEvent

Field	Syntax	Description
id	protected int id	ID of the source XSLTransformer object

Methods of XSLTransformerEvent

Table 10–17 Summary of Methods of XSLTransformerEvent

Method	Description
XSLTransformerEvent() on page 10-37	Constructs the XSLTransformerEvent object using the XSLTransformer source object and its unique id.
getID() on page 10-37	Returns unique id of the XSLTransformer object

XSLTransformerEvent()

Syntax

Constructs the XSLTransformerEvent object using the XSLTransformer source object and its unique id.

Description

```
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()

Syntax

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.

Description

```
public int getID();
```

XSLTransformerListener Interface

Syntax of XSLTransformerListener

```
public interface XSLTransformerListener extends java.util.EventListener
```

Description of XSLTransformerListener

This interface must be implemented in order to receive notifications about events during the asynchronous transformation. The class implementing this interface must be added to the XSLTransformer using addXSLTransformerListener method.

Methods of XSLTransformerListener

Table 10–18 Summary of Methods of XSLTransformerListener

Method	Description
xslTransformerError() on page 10-38	This method is called when parse or transformation error occur.
xslTransformerOver() on page 10-39	This method is called when the transformation is complete.
xslTransformerStarted() on page 10-39	This method is called when the transformation starts.

xslTransformerError()

Description

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()

Description

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()

Description

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

Description

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 10–19.

Table 10–19 Summary of Classes of oracle.xml.dbviewer

Class	Description
DBViewer Class on page 10-41	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 10-67	This class provides information about the DBViewer Bean.

DBViewer Class

Description of DBViewer

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 of DBViewer

```
public class DBViewer extends javax.swing.JPanel implements java.io.Serializable

java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--javax.swing.JComponent
                    |
                    +--javax.swing.JPanel
                          |
                          +--oracle.xml.dbviewer.DBViewer
```

Implemented Interfaces of DBViewer

- `javax.accessibility.Accessible`
- `java.awt.image.ImageObserver`
- `java.awt.MenuContainer`
- `java.io.Serializable`

Methods of DBViewer

Table 10–20 Summary of Methods of DBViewer

Method	Description
DBViewer() on page 10-45	Constructs a new instance of DBViewer.
getHostname() on page 10-45	Retrieves the database host name.
getInstancename() on page 10-45	Retrieves the database instance name.
getPassword() on page 10-45	Retrieves the user password.
getPort() on page 10-45	Retrieves the database port number as a String.
getResBuffer() on page 10-46	Retrieves the content of the result buffer.
getResCLOBFileName() on page 10-46	Retrieves the result CLOB file name.
getResCLOBTableName() on page 10-46	Retrieves the result CLOB table name.
getResFileName() on page 10-46	Retrieves the result file name.
getUsername() on page 10-47	Retrieves the user name.
getXmlBuffer() on page 10-47	Retrieves the content of the XML buffer.
getXmlCLOBFileName() on page 10-47	Retrieves the XML CLOB file name.
getXmlCLOBTableName() on page 10-47	Retrieves the XML CLOB table name.
getXmlFileName() on page 10-47	Retrieves the XML file name.
getXMLStringFromSQL() on page 10-48	Retrieves the XML presentation of a result set from SQL query as an XMLString.
getXslBuffer() on page 10-48	Retrieves the content of the XSL buffer.
getXslCLOBFileName() on page 10-48	Retrieves the XSL CLOB file name.
getXslCLOBTableName() on page 10-48	Retrieves XSL CLOB table name.
getXslFileName() on page 10-49	Retrieves XSL file name.
loadResBuffer() on page 10-49	Loads the result buffer.
loadResBufferFromClob() on page 10-50	Loads the result buffer from CLOB file.
loadResBufferFromFile() on page 10-50	Loads the result buffer from file.
loadXmlBuffer() on page 10-50	Loads the XML buffer.
loadXmlBufferFromClob() on page 10-51	Loads the XML buffer from CLOB file.
loadXmlBufferFromFile() on page 10-51	Loads the XML buffer from file.

Table 10–20 Summary of Methods of DBViewer (Cont.)

Method	Description
loadXMLBufferFromSQL() on page 10-51	Loads the XML buffer from SQL result set.
loadXslBuffer() on page 10-52	Loads the XSL buffer from file.
loadXslBufferFromClob() on page 10-52	Loads the XSL buffer from CLOB file.
loadXslBufferFromFile() on page 10-52	Loads the XSL buffer from file.
parseResBuffer() on page 10-53	Parses the result buffer, refreshes the tree and source views, and returns the XMLDocument.
parseXmlBuffer() on page 10-53	Parse the XML buffer, refreshes the tree and source views, and returns the XMLDocument.
parseXslBuffer() on page 10-53	Parses the XSL buffer, refreshes the tree and source views, and returns the XMLDocument.
saveResBuffer() on page 10-53	Saves the result buffer to file.
saveResBufferToClob() on page 10-54	Saves the result buffer to CLOB file.
saveResBufferToFile() on page 10-54	Saves the result buffer to file.
saveXmlBuffer() on page 10-54	Saves the XML buffer to file.
saveXmlBufferToClob() on page 10-55	Saves the XML buffer to CLOB file.
saveXmlBufferToFile() on page 10-55	Saves the XML buffer to file.
saveXslBuffer() on page 10-55	Saves the XSL buffer to file.
saveXslBufferToClob() on page 10-56	Saves the XSL buffer to CLOB file.
saveXslBufferToFile() on page 10-56	Saves the XSL buffer to file.
setHostname() on page 10-56	Sets database host name.
setInstancename() on page 10-57	Sets database instance name.
setPassword() on page 10-57	Sets user password.
setPort() on page 10-57	Sets database port number.
setResBuffer() on page 10-58	Sets new text in the result buffer.
setResCLOBFileName() on page 10-58	Sets result CLOB file name.
setResCLOBTableName() on page 10-58	Sets result CLOB table name.
setResFileName() on page 10-59	Sets result file name.
setResHtmlView() on page 10-59	Show the result buffer as rendered HTML.

Table 10–20 Summary of Methods of DBViewer (Cont.)

Method	Description
setResSourceEditView() on page 10-60	Shows the result buffer as XML source and enter edit mode.
setResSourceView() on page 10-60	Shows the result buffer as XML source.
setResTreeView() on page 10-60	Shows the result buffer as an XML tree view.
setUsername() on page 10-61	Sets the user name.
setXmlBuffer() on page 10-61	Sets new text in the XML buffer.
setXmlCLOBFileName() on page 10-61	Sets XML CLOB file name.
setXmlCLOBTableName() on page 10-62	Sets the XML CLOB table name.
setXmlFileName() on page 10-62	Sets the XML file name.
setXmlSourceEditView() on page 10-62	Shows the XML buffer as XML source and enter edit mode.
setXmlSourceView() on page 10-63	Shows the XML buffer as an XML source.
setXmlTreeView() on page 10-63	Shows the XML buffer as tree.
setXslBuffer() on page 10-64	Sets new text in the XSL buffer.
setXslCLOBFileName() on page 10-64	Sets the XSL CLOB file name.
setXslCLOBTableName() on page 10-64	Sets the XSL CLOB table name.
setXslFileName() on page 10-65	Sets the XSL file name.
setXslSourceEditView() on page 10-65	Shows the XSL buffer as XML source and enter edit mode.
setXslSourceView() on page 10-65	Shows the XSL buffer as an XML source.
setXslTreeView() on page 10-66	Shows the XSL buffer as tree.
transformToDoc() on page 10-66	Transforms the content of the XML buffer by applying the stylesheet from the XSL buffer.
transformToRes() on page 10-66	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 10-66	Transforms the content of the XML buffer by applying the stylesheet from the XSL buffer.

DBViewer()

Description

Constructs a new instance of DBViewer.

Syntax

```
public  DBViewer();
```

getHostname()

Description

Retrieves the database host name.

Syntax

```
public java.lang.String getHostname();
```

getInstancename()

Description

Retrieves the database instance name.

Syntax

```
public java.lang.String getInstancename();
```

getPassword()

Description

Retrieves the user password.

Syntax

```
public java.lang.String getPassword();
```

getPort()

Description

Retrieves the database port number as a String.

Syntax

```
public java.lang.String getPort();
```

getResBuffer()

Description

Retrieves the content of the result buffer.

Syntax

```
public java.lang.String getResBuffer();
```

getResCLOBFileName()

Description

Retrieves the result CLOB file name.

Syntax

```
public java.lang.String getResCLOBFileName();
```

getResCLOBTableName()

Description

Retrieves the result CLOB table name.

Syntax

```
public java.lang.String getResCLOBTableName();
```

getResFileName()

Description

Retrieves the result file name.

Syntax

```
public java.lang.String getResFileName();
```

getUsername()

Description
Retrieves the user name.

Syntax
```
public java.lang.String getUsername();
```

getXmlBuffer()

Description
Retrieves the content of the XML buffer.

Syntax
```
public java.lang.String getXmlBuffer();
```

getXmlCLOBFileName()

Description
Retrieves the XML CLOB file name.

Syntax
```
public java.lang.String getXmlCLOBFileName();
```

getXmlCLOBTableName()

Description
Retrieves the XML CLOB table name.

Syntax
```
public java.lang.String getXmlCLOBTableName();
```

getXmlFileName()

Description
Retrieves the XML file name.

Syntax

```
public java.lang.String getXmlFileName();
```

getXMLStringFromSQL()

Description

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()

Description

Retrieves the content of the XSL buffer.

Syntax

```
public java.lang.String getXslBuffer();
```

getXslCLOBFileName()

Description

Retrieves the XSL CLOB file name.

Syntax

```
public java.lang.String getXslCLOBFileName();
```

getXslCLOBTableName()

Description

Retrieves XSL CLOB table name.

Syntax

```
public java.lang.String getXslCLOBTableName();
```

getXslFileName()

Description

Retrieves XSL file name.

Syntax

```
public java.lang.String getXslFileName();
```

loadResBuffer()

Description

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()

Description

Loads the result buffer from CLOB file.

Syntax

```
public void loadResBufferFromClob();
```

loadResBufferFromFile()

Description

Loads the result buffer from file.

Syntax

```
public void loadResBufferFromFile();
```

loadXmlBuffer()

Description

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.
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.

Parameter	Description
clobTablename	The CLOB table name.
clobFilename	The CLOB file name.
resDoc	The XMLDocument.

loadXmlBufferFromClob()

Description
Loads the XML buffer from CLOB file.

Syntax
```
public void loadXmlBufferFromClob();
```

loadXmlBufferFromFile()

Description
Loads the XML buffer from file.

Syntax
```
public void loadXmlBufferFromFile();
```

loadXMLBufferFromSQL()

Description
Loads the XML buffer from SQL result set.

Syntax
```
public void loadXMLBufferFromSQL( String sqlText);
```

Parameter	Description
sqlText	SQL text

loadXslBuffer()

Description

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()

Description

Loads the XSL buffer from CLOB file.

Syntax

```
public void loadXslBufferFromClob();
```

loadXslBufferFromFile()

Description

Loads the XSL buffer from file.

Syntax

```
public void loadXslBufferFromFile();
```

parseResBuffer()

Description

Parses the result buffer, refreshes the tree and source views, and returns the XMLDocument.

Syntax

```
public oracle.xml.parser.v2.XMLDocument parseResBuffer();
```

parseXmlBuffer()

Description

Parse the XML buffer, refreshes the tree and source views, and returns the XMLDocument.

Syntax

```
public oracle.xml.parser.v2.XMLDocument parseXmlBuffer();
```

parseXslBuffer()

Description

Parses the XSL buffer, refreshes the tree and source views, and returns the XMLDocument.

Syntax

```
public oracle.xml.parser.v2.XMLDocument parseXslBuffer();
```

saveResBuffer()

Description

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()

Description
Saves the result buffer to CLOB file.

Syntax
```
public void saveResBufferToClob();
```

saveResBufferToFile()

Description
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()

Description
Saves the XML buffer to CLOB file.

Syntax
```
public void saveXmlBufferToClob();
```

saveXmlBufferToFile()

Description
Saves the XML buffer to file.

Syntax
```
public void saveXmlBufferToFile();
```

saveXslBuffer()

Description
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()

Description
Saves the XSL buffer to CLOB file.

Syntax
```
public void saveXslBufferToClob();
```

saveXslBufferToFile()

Description
Saves the XSL buffer to file.

Syntax
```
public void saveXslBufferToFile();
```

setHostname()

Description
Sets database host name.

Syntax

```
public void setHostname( java.lang.String hostname);
```

Parameter	Description
hostname	The host name.

setInstancename()

Description

Sets database instance name.

Syntax

```
public void setInstancename( String instancename);
```

Parameter	Description
instancename	The database instance name.

setPassword()

Description

Sets user password.

Syntax

```
public void setPassword( String password);
```

Parameter	Description
password	The user password.

setPort()

Description

Sets database port number.

Syntax

```
public void setPort( String port);
```

Parameter	Description
port	String containing the port number.

setResBuffer()

Description

Sets new text in the result buffer.

Syntax

```
public void setResBuffer( String text);
```

Parameter	Description
text	The new text.

setResCLOBFileName()

Description

Sets result CLOB file name.

Syntax

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

Parameter	Description
name	Result CLOB file name.

setResCLOBTableName()

Description

Sets result CLOB table name.

Syntax

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

Parameter	Description
name	Result CLOB table name.

setResFileName()

Description

Sets result file name.

Syntax

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

Parameter	Description
name	Result file name.

setResHtmlView()

Description

Show the result buffer as rendered HTML.

Syntax

```
public void setResHtmlView( boolean on);
```

Parameter	Description
on	Switch for showing result buffer as HTML; TRUE for showing, FALSE otherwise.

setResSourceEditView()

Description

Shows the result buffer as XML source and enter edit mode.

Syntax

```
public void setResSourceEditView( boolean on);
```

Parameter	Description
on	Switch for showing result buffer as HTML with edit mode; TRUE for showing, FALSE otherwise.

setResSourceView()

Description

Shows the result buffer as XML source.

Syntax

```
public void setResSourceView( boolean on);
```

Parameter	Description
on	Switch for showing result buffer as XML source; TRUE for showing, FALSE otherwise.

setResTreeView()

Description

Shows the result buffer as an XML tree view.

Syntax

```
public void setResTreeView( boolean on);
```

Parameter	Description
on	Switch for showing result buffer as an XML tree view; TRUE for showing, FALSE otherwise.

setUsername()

Description

Sets the user name.

Syntax

```
public void setUsername( String username);
```

Parameter	Description
username	The user name.

setXmlBuffer()

Description

Sets new text in the XML buffer.

Syntax

```
public void setXmlBuffer( String text)
```

Parameter	Description
text	The XML text

setXmlCLOBFileName()

Description

Sets XML CLOB file name.

Syntax

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

Parameter	Description
name	The XML CLOB file name.

setXmlCLOBTableName()

Description
Sets the XML CLOB table name.

Syntax
```
public void setXmlCLOBTableName( String name);
```

Parameter	Description
name	The XML CLOB table name.

setXmlFileName()

Description
Sets the XML file name.

Syntax
```
public void setXmlFileName( String name);
```

Parameter	Description
name	The XML file name.

setXmlSourceEditView()

Description
Shows the XML buffer as XML source and enter edit mode.

Syntax

```
public void setXmlSourceEditView( boolean on);
```

Parameter	Description
`on`	Switch for showing the XML buffer as XML source with edit mode; TRUE for showing, FALSE otherwise.

setXmlSourceView()

Description

Shows the XML buffer as an XML source.

Syntax

```
public void setXmlSourceView( boolean on);
```

Parameter	Description
`on`	Switch for showing the XML buffer as XML source; TRUE for showing, FALSE otherwise.

setXmlTreeView()

Description

Shows the XML buffer as tree.

Syntax

```
public void setXmlTreeView( boolean on);
```

Parameter	Description
`on`	Switch for showing the XML buffer as XML tree; TRUE for showing, FALSE otherwise.

setXslBuffer()

Description

Sets new text in the XSL buffer.

Syntax

```
public void setXslBuffer( String text);
```

Parameter	Description
text	The XSL text.

setXslCLOBFileName()

Description

Sets the XSL CLOB file name.

Syntax

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

Parameter	Description
name	The XSL CLOB file name.

setXslCLOBTableName()

Description

Sets the XSL CLOB table name.

Syntax

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

Parameter	Description
name	The XSL CLOB table name.

setXslFileName()

Description

Sets the XSL file name.

Syntax

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

Parameter	Description
name	The XSL file name.

setXslSourceEditView()

Description

Shows the XSL buffer as XML source and enter edit mode.

Syntax

```
public void setXslSourceEditView( boolean on);
```

Parameter	Description
on	Switch for showing the XSL buffer as XML source with edit mode; TRUE for showing, FALSE otherwise.

setXslSourceView()

Description

Shows the XSL buffer as an XML source.

Syntax

```
public void setXslSourceView( boolean on);
```

Parameter	Description
on	Switch for showing the XSL buffer as XML source; TRUE for showing, FALSE otherwise.

setXslTreeView()

Description

Shows the XSL buffer as tree.

Syntax

```
public void setXslTreeView( boolean on);
```

Parameter	Description
on	Switch for showing the XSL buffer as XML tree; TRUE for showing, FALSE otherwise.

transformToDoc()

Description

Transforms the content of the XML buffer by applying the stylesheet from the XSL buffer.

Syntax

```
public oracle.xml.parser.v2.XMLDocument transformToDoc();
```

transformToRes()

Description

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()

Description

Transfroms the content of the XML buffer by applying the stylesheet from the XSL buffer.

Syntax

```
public java.lang.String transformToString();
```

DBViewerBeanInfo Class

Description of DBViewerBeanInfo

This class provides information about the DBViewer Bean.

Syntax of DBViewerBeanInfo

```
public class DBViewerBeanInfo extends java.beans.SimpleBeanInfo

java.lang.Object
  |
  +--java.beans.SimpleBeanInfo
        |
        +--oracle.xml.dbviewer.DBViewerBeanInfo
```

Implemented Interfaces of DBViewerBeanInfo

- `java.beans.BeanInfo`

Methods of DBViewerBeanInfo

Table 10–21 Summary of Methods of DBViewerBeanInfo

Method	Description
DBViewerBeanInfo() on page 10-67	Class constructor.
getIcon() on page 10-68	Retrieves an image object representing the requested icon type for DBViewer bean in toolbars, toolboxes, and so on.
getPropertyDescriptors() on page 10-68	Retrieves an array of DBViwer bean's editable PropertyDescriptors.

DBViewerBeanInfo()

Description

Class constructor.

Syntax

```
public  DBViewerBeanInfo();
```

getIcon()

Description

Retrieves an image object representing the requested icon type for DBViewer 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()

Description

Retrieves an array of DBViwer bean's editable PropertyDescriptors. Overrides `getPropertyDescriptors()` in `java.beans.SimpleBeanInfo` Class.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

oracle.xml.srcviewer Package

Description

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 10–22.

Table 10–22 Summary of Classes of oracle.xml.srcviewer

Class	Description
XMLSourceView Class on page 10-70	Shows an XML document.
XMLSourceViewBeanInfo Class on page 10-88	This class provides information about the XMLSourceView Bean.

XMLSourceView Class

Description of XMLSourceView

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 of XMLSourceView

```
public class XMLSourceView extends javax.swing.JPanel implements
java.io.Serializable

java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--javax.swing.JComponent
                    |
                    +--javax.swing.JPanel
                          |
                          +--oracle.xml.srcviewer.XMLSourceView
```

Implemented Interfaces of XMLSourceView

- `javax.accessibility.Accessible`
- `java.awt.image.ImageObserver`
- `java.awt.MenuContainer`
- `java.io.Serializable`

Fields of XMLSourceView

Table 10–23 Fields of ElementDecl

Field	Syntax	Description
inputDOMDocument	protected org.w3c.dom.Document inputDOMDocument	XML Document to be displayed.

Table 10–23 Fields of ElementDecl

Field	Syntax	Description
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.

Methods of XMLSourceView

Table 10–24 Summary of Methods of XMLSourceView

Method	Description
XMLSourceView() on page 10-73	The class constructor. Creates an object of type XMLSourceView.
fontGet() on page 10-73	Extracts and returns the font from a given attribute set.
fontSet() on page 10-74	Sets the mutable attribute set font.
getAttributeNameFont() on page 10-74	Returns the Attribute Value font.
getAttributeNameForeground() on page 10-74	Returns the Attribute Name foreground color.
getAttributeValueFont() on page 10-75	Returns the Attribute Value font.
getAttributeValueForeground() on page 10-75	Returns the Attribute Value foreground color.
getBackground() on page 10-75	Returns the background color.
getCDATAFont() on page 10-75	Returns the CDATA font.
getCDATAForeground() on page 10-75	Returns the CDATA foreground color.
;getCommentDataFont() on page 10-76	Returns the Comment Data font.
getCommentDataForeground() on page 10-76	Returns the Comment Data foreground color.
getEditedText() on page 10-76	Returns the edited text.

Table 10–24 Summary of Methods of XMLSourceView (Cont.)

Method	Description
getJTextPane() on page 10-76	Returns the viewer `JTextPane` component used by XMLSourceViewer.
getMinimumSize() on page 10-77	Returns the XMLSourceView minimal size.
getNodeAtOffset() on page 10-77	Returns the XML node at a given offset.
getPCDATAFont() on page 10-77	Returns the PCDATA font.
getPCDATAForeground() on page 10-77	Returns the PCDATA foreground color.
getPIDataFont() on page 10-78	Returns the PI Data font.
getPIDataForeground() on page 10-78	Returns the PI Data foreground color.
getPINameFont() on page 10-78	Returns the PI Name font.
getPINameForeground() on page 10-78	Returns the PI Data foreground color.
getSymbolFont() on page 10-78	Returns the NOTATION Symbol font.
getSymbolForeground() on page 10-79	Returns the NOTATION Symbol foreground color.
getTagFont() on page 10-79	Returns the Tag font.
getTagForeground() on page 10-79	Returns the Tag foreground color.
getText() on page 10-79	Returns the XML document as a String.
isEditable() on page 10-80	Returns boolean to indicate whether this object is editable.
selectNodeAt() on page 10-80	Moves the cursor to XML Node at specified offset.
setAttributeNameFont() on page 10-80	Sets the Attribute Name font.
setAttributeNameForeground() on page 10-81	Sets the Attribute Name foreground color.
setAttributeValueFont() on page 10-81	Sets the Attribute Value font.
setAttributeValueForeground() on page 10-81	Sets the Attribute Value foreground color.
setBackground() on page 10-82	Sets the background color.
setCDATAFont() on page 10-82	Sets the CDATA font.
setCDATAForeground() on page 10-82	Sets the CDATA foreground color.
setCommentDataFont() on page 10-83	Sets the Comment font.
setCommentDataForeground() on page 10-83	Sets the Comment foreground color.

Table 10–24 Summary of Methods of XMLSourceView (Cont.)

Method	Description
setEditable() on page 10-83	Sets the specified boolean to indicate whether this object should be editable.
setPCDATAFont() on page 10-84	Sets the PCDATA font.
setPCDATAForeground() on page 10-84	Sets the PCDATA foreground color.
setPIDataFont() on page 10-84	Sets the PI Data font.
setPIDataForeground() on page 10-85	Sets the PI Data foreground color.
setPINameFont() on page 10-85	Sets the PI Name font.
setPINameForeground() on page 10-85	Sets the PI Name foreground color.
setSelectedNode() on page 10-86	Sets the cursor position at the selected XML node.
setSymbolFont() on page 10-86	Sets the NOTATION Symbol font.
setSymbolForeground() on page 10-86	Sets the NOTATION Symbol foreground color.
setTagFont() on page 10-87	Sets the Tag font.
setTagForeground() on page 10-87	Sets the Tag foreground color.
setXMLDocument() on page 10-87	Associates the XMLviewer with a XML document.

XMLSourceView()

Description
The class constructor. Creates an object of type XMLSourceView.

Syntax
```
public  XMLSourceView();
```

fontGet()

Description
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()

Description

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()

Description

Returns the Attribute Value font.

Syntax

```
public java.awt.Font getAttributeNameFont();
```

getAttributeNameForeground()

Description

Returns the Attribute Name foreground color.

Syntax

```
public java.awt.Color getAttributeNameForeground();
```

getAttributeValueFont()

Description

Returns the Attribute Value font.

Syntax

```
public java.awt.Font getAttributeValueFont();
```

getAttributeValueForeground()

Description

Returns the Attribute Value foreground color.

Syntax

```
public java.awt.Color getAttributeValueForeground();
```

getBackground()

Description

Returns the background color. Overrides getBackground() in java.awt.Component class.

Syntax

```
public java.awt.Color getBackground();
```

getCDATAFont()

Description

Returns the CDATA font.

Syntax

```
public java.awt.Font getCDATAFont();
```

getCDATAForeground()

Description

Returns the CDATA foreground color.

Syntax

```
public java.awt.Color getCDATAForeground();
```

;getCommentDataFont()

Description

Returns the Comment Data font.

Syntax

```
public java.awt.Font getCommentDataFont();
```

getCommentDataForeground()

Description

Returns the Comment Data foreground color.

Syntax

```
public java.awt.Color getCommentDataForeground();
```

getEditedText()

Description

Returns the edited text.

Syntax

```
public java.lang.String getEditedText();
```

getJTextPane()

Description

Returns the viewer `JTextPane` component used by XMLSourceViewer.

Syntax

```
public javax.swing.JTextPane getJTextPane();
```

getMinimumSize()

Description

Returns the XMLSourceView minimal size. Overrides getMinimumSize() in javax.swing.JComponent class.

Syntax

```
public java.awt.Dimension getMinimumSize();
```

getNodeAtOffset()

Description

Returns the XML node at a given offset.

Syntax

```
public org.w3c.dom.Node getNodeAtOffset( int i);
```

Parameter	Description
i	The node offset.

getPCDATAFont()

Description

Returns the PCDATA font.

Syntax

```
public java.awt.Font getPCDATAFont();
```

getPCDATAForeground()

Description

Returns the PCDATA foreground color.

Syntax

```
public java.awt.Color getPCDATAForeground();
```

getPIDataFont()

Description
Returns the PI Data font.

Syntax
```
public java.awt.Font getPIDataFont();
```

getPIDataForeground()

Description
Returns the PI Data foreground color.

Syntax
```
public java.awt.Color getPIDataForeground();
```

getPINameFont()

Description
Returns the PI Name font.

Syntax
```
public java.awt.Font getPINameFont();
```

getPINameForeground()

Description
Returns the PI Data foreground color.

Syntax
```
public java.awt.Color getPINameForeground();
```

getSymbolFont()

Description
Returns the NOTATION Symbol font.

Syntax

```
public java.awt.Font getSymbolFont();
```

getSymbolForeground()

Description

Returns the NOTATION Symbol foreground color.

Syntax

```
public java.awt.Color getSymbolForeground();
```

getTagFont()

Description

Returns the Tag font.

Syntax

```
public java.awt.Font getTagFont();
```

getTagForeground()

Description

Returns the Tag foreground color.

Syntax

```
public java.awt.Color getTagForeground();
```

getText()

Description

Returns the XML document as a String.

Syntax

```
public java.lang.String getText();
```

isEditable()

Description

Returns boolean to indicate whether this object is editable.

Syntax

```
public boolean isEditable();
```

selectNodeAt()

Description

Moves the cursor to XML Node at the specified offset.

Syntax

```
public void selectNodeAt( int i);
```

Parameter	Description
i	The node offset.

setAttributeNameFont()

Description

Sets the Attribute Name font.

Syntax

```
public void setAttributeNameFont( java.awt.Font font);
```

Parameter	Description
font	The new Font Attribute Name.

setAttributeNameForeground()

Description

Sets the Attribute Name foreground color.

Syntax

```
public void setAttributeNameForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for Attribute Name.

setAttributeValueFont()

Description

Sets the Attribute Value font.

Syntax

```
public void setAttributeValueFont( java.awt.Font font);
```

Parameter	Description
font	The new Font Attribute Value

setAttributeValueForeground()

Description

Sets the Attribute Value foreground color.

Syntax

```
public void setAttributeValueForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for Attribute Value.

setBackground()

Description

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()

Description

Sets the CDATA font.

Syntax

```
public void setCDATAFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for CDATA.

setCDATAForeground()

Description

Sets the CDATA foreground color.

Syntax

```
public void setCDATAForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for CDATA.

setCommentDataFont()

Description

Sets the Comment font.

Syntax

```
public void setCommentDataFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for the XML Comments.

setCommentDataForeground()

Description

Sets the Comment foreground color.

Syntax

```
public void setCommentDataForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for Comment.

setEditable()

Description

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()

Description

Sets the PCDATA font.

Syntax

```
public void setPCDATAFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for PCDATA.

setPCDATAForeground()

Description

Sets the PCDATA foreground color.

Syntax

```
public void setPCDATAForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for PCDATA.

setPIDataFont()

Description

Sets the PI Data font.

Syntax

```
public void setPIDataFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for PI Data.

setPIDataForeground()

Description

Sets the PI Data foreground color.

Syntax

```
public void setPIDataForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for PI Data.

setPINameFont()

Description

Sets the PI Name font.

Syntax

```
public void setPINameFont(java.awt.Font font);
```

Parameter	Description
font	The new Font for the PI Names.

setPINameForeground()

Description

Sets the PI Name foreground color.

Syntax

```
public void setPINameForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for PI Name.

setSelectedNode()

Description

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()

Description

Sets the NOTATION Symbol font.

Syntax

```
public void setSymbolFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for NOTATION Symbol.

setSymbolForeground()

Description

Sets the NOTATION Symbol foreground color.

Syntax

```
public void setSymbolForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for NOTATION Symbol.

setTagFont()

Description

Sets the Tag font.

Syntax

```
public void setTagFont( java.awt.Font font);
```

Parameter	Description
font	The new Font for the XML Tags.

setTagForeground()

Description

Sets the Tag foreground color.

Syntax

```
public void setTagForeground( java.awt.Color color);
```

Parameter	Description
color	The new Color for the XML Tags.

setXMLDocument()

Description

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

Derscription of XMLSourceViewBeanInfo

This class provides information about the XMLSourceView Bean.

Syntax of XMLSourceViewBeanInfo

```
public class XMLSourceViewBeanInfo extends java.beans.SimpleBeanInfo

java.lang.Object
  |
  +--java.beans.SimpleBeanInfo
        |
        +--oracle.xml.srcviewer.XMLSourceViewBeanInfo
```

Implemented Interfaces of XMLSourceViewBeanInfo

```
java.beans.BeanInfo
```

Methods of XMLSourceViewBeanInfo

Table 10–25 Summary of Methods of XMLSourceViewBeanInfo

Method	Description
XMLSourceViewBeanInfo() on page 10-88	Class constructor.
getIcon() on page 10-89	Retrieves an image object representing the requested icon type for XMLSourceView bean in toolbars, toolboxes, and so on.
getPropertyDescriptors() on page 10-89	Retrieves an array of XMLSourveView bean's editable PropertyDescriptors.

XMLSourceViewBeanInfo()

Description

Class constructor.

Syntax

```
public  XMLSourceViewBeanInfo();
```

getIcon()

Description

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()

Description

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

Description

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 10–26.

Table 10–26 Summary of Classes of oracle.xml.transviewer

Class	Description
DBAccess Class on page 10-91	Maintains CLOB tables that can hold multiple XML and text documents.
DBAccessBeanInfo Class on page 10-100	This class provides information about the DBAccess Bean.
XMLTransformPanel Class on page 10-102	Applies XSL transformations to XML documents and visualizes the result.
XMLTransformPanelBeanInfo Class on page 10-103	This class provides information about the XMLTransformPanel Bean
XMLTransViewer Class on page 10-105	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

Description of DBAccess

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 of DBAccess

```
public class DBAccess extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.transviewer.DBAccess
```

Methods of DBAccess

Table 10–27 Summary of Methods of DBAcceess

Method	Description
DBAccess() on page 10-92	Class constructor.
createBLOBTable() on page 10-92	Create BLOB table, returning TRUE if successful.
createXMLTable() on page 10-93	Creates XML table, returning TRUE if successful.
deleteBLOBName() on page 10-93	Deletes binary file from BLOB table, returning TRUE if successful.
deleteXMLName() on page 10-94	Delete file from XML table, returning TRUE if successful.

Table 10–27 Summary of Methods of DBAcceess (Cont.)

Method	**Description**
dropBLOBTable() on page 10-94	Deletes BLOB table. TRUE if successful.
dropXMLTable() on page 10-95	Deletes XML table. Returns TRUE if successful.
getBLOBData() on page 10-95	Retrieves binary file from BLOB table as a byte array. Returns TRUE if successful.
getNameSize() on page 10-95	Returns the size of the field where the filename is kept.
getXMLData() on page 10-96	Retrieves text file from XML table as a String.
getXMLNames() on page 10-96	Returns all file names in XML table as a String array.
getXMLTableNames() on page 10-97	Retrieves an array of all XML table names, starting with a user-specified string.
insertBLOBData() on page 10-97	Inserts binary file as a row in BLOB table. Returns TRUE if successful.
insertXMLData() on page 10-98	Inserts text file as a row in XML table. Returns TRUE if successful.
isXMLTable() on page 10-98	Checks if the table is an XML table, in which case returns TRUE.
replaceXMLData() on page 10-99	Replaces text file as a row in XML table, returning TRUE if successful.
xmlTableExists() on page 10-99	Checks if the XML table exists, returning TRUE if successful.

DBAccess()

Description

Class constructor.

Syntax

```
public  DBAccess();
```

createBLOBTable()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Returns the size of the field where the filename is kept.

Syntax

```
public int getNameSize();
```

getXMLData()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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

Description of DBAccessBeanInfo

This class provides information about the DBAccess Bean.

Syntax of DBAccessBeanInfo

```
public class DBAccessBeanInfo extends java.beans.SimpleBeanInfo

java.lang.Object
  |
  +--java.beans.SimpleBeanInfo
        |
        +--oracle.xml.transviewer.DBAccessBeanInfo
```

Implemented Interfaces of DBAccessBeanInfo

- `java.beans.BeanInfo`

Methods of DBAccessBeanInfo

Table 10–28 Summary of Methods of DBAccessBeanInfo

Method	Description
DBAccessBeanInfo() on page 10-100	Class constructor.
getIcon() on page 10-101	Retrieves an image object representing the requested icon type for DBAcceess bean in toolbars, toolboxes, and so on.
getPropertyDescriptors() on page 10-101	Retrieves an array of DBAccess bean's editable PropertyDescriptors.

DBAccessBeanInfo()

Description

Class constructor.

Syntax

```
public  DBAccessBeanInfo();
```

getIcon()

Description

Retrieves an image object representing the requested icon type for DBAcceess 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()

Description

Retrieves an array of DBAccess bean's editable PropertyDescriptors. Overrides `getPropertyDescriptors()` in `java.beans.SimpleBeanInfo` Class.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

XMLTransformPanel Class

Description of XMLTransformPanel

XMLTransformPanel visual bean. Applies XSL transformations to XML documents and visualizes the result. Allows editing of input XML and XSL documents and files.

Syntax of XMLTransformPanel

```
public class XMLTransformPanel extends javax.swing.JPanel

java.lang.Object
  |
  +--java.awt.Component
      |
      +--java.awt.Container
          |
          +--javax.swing.JComponent
              |
              +--javax.swing.JPanel
                  |
                  +--oracle.xml.transviewer.XMLTransformPanel
```

Implemented Interfaces of XMLTransformPanel

- `javax.accessibility.Accessible`
- `java.awt.image.ImageObserver`
- `java.awt.MenuContainer`
- `java.io.Serializable`

Methods of Description of XMLTransformPanel

XMLTransformPanel()

Description

The class constructor. Creates an object of type `XMLTransformPanel`.

Syntax

```
public  XMLTransformPanel();
```

XMLTransformPanelBeanInfo Class

Description of XMLTransformPanelBeanInfo

This class provides information about the XMLTransformPanel Bean.

Syntax of XMLTransformPanelBeanInfo

```
public class XMLTransformPanelBeanInfo extends java.beans.SimpleBeanInfo

java.lang.Object
  |
  +--java.beans.SimpleBeanInfo
        |
        +--oracle.xml.transviewer.XMLTransformPanelBeanInfo
```

Implemented Interfaces of XMLTransformPanelBeanInfo

- `java.beans.BeanInfo`

Methods of XMLTransformPanelBeanInfo

Table 10–29 Summary of Methods of XMLTransformPanelBeanInfo

Method	Description
XMLTransformPanelBeanInfo() on page 10-103	Class constructor.
getIcon() on page 10-104	Retrieves an image object representing the requested icon type for XMLTransformPanel bean in toolbars, toolboxes, and so on.
getPropertyDescriptors() on page 10-104	Retrieves an array of XMLTransformPanel bean's editable PropertyDescriptors.

XMLTransformPanelBeanInfo()

Description

Class constructor.

Syntax

```
public  XMLTransformPanelBeanInfo();
```

getIcon()

Description

Retrieves an image object representing the requested icon type for XMLTransformPanel 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()

Description

Retrieves an array of XMLTransformPanel bean's editable PropertyDescriptors. Overrides `getPropertyDescriptors()` in `java.beans.SimpleBeanInfo` Class.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

XMLTransViewer Class

Description of XMLTransViewer

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 of XMLTransViewer

```
public class XMLTransViewer extends java.lang.Object

java.lang.Object
  |
  +--oracle.xml.transviewer.XMLTransViewer
```

Methods of XMLTransViewer

Table 10–30 Summary of Methods of XMLTransViewer

Method	Description
XMLTransViewer() on page 10-105	Class constructor
getReleaseVersion() on page 10-105	Returns the release version of the Oracle XML Transviewer, as a String.
main() on page 10-106	Starts a new XMLTransViewer.

XMLTransViewer()

Description

Class constructor

Syntax

```
public  XMLTransViewer();
```

getReleaseVersion()

Description

Returns the release version of the Oracle XML Transviewer, as a String.

Syntax

```
public static java.lang.String getReleaseVersion();
```

main()

Description

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

Description

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 10–31.

Table 10–31 Summary of Classes of oracle.xml.transviewer

Class	Description
XMLTreeView Class on page 10-108	Shows an XML document as a tree.
XMLTreeViewBeanInfo Class on page 10-111	This class provides information about the XMLTreeView Bean.

XMLTreeView Class

Description of XMLTreeView

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 of XMLTreeView

```
public class XMLTreeView extends javax.swing.JPanel

java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--javax.swing.JComponent
                    |
                    +--javax.swing.JPanel
                          |
                          +--oracle.xml.treeviewer.XMLTreeView
```

Implemented Interfaces of XMLTreeView

`javax.accessibility.Accessible, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable`

Fields of XMLTreeView of XMLTreeView

Table 10–32 Fields of XMLTreeView

Field	Syntax	Description
model	protected oracle.xml.treeviewer.XMLTreeModel model	Data model for JTree using information from DOM nodes.
scrollPane	protected transient javax.swing.JScrollPane scrollPane	Container used to display the Tree. Supports horizontal and vertical scrolling.

Table 10–32 Fields of XMLTreeView (Cont.)

Field	Syntax	Description
theTree	protected transient javax.swing.JTree theTree	Java component to render the various nodes of the DOM Tree.

Methods of XMLTreeView

Table 10–33 Summary of Methods of XMLTreeView

Method	Description
XMLTreeView() on page 10-109	The class constructor.
getPreferredSize() on page 10-109	Returns the XMLTreeView preferred size.
getTree() on page 10-110	Returns the visual tree as a JTree.
getXMLTreeModel() on page 10-110	Returns the datamodel for JTree as XMLTreeModel.
setXMLDocument() on page 10-110	Associates the XMLTreeViewer with a XML document.
updateUI() on page 10-110	Forces the XMLTreeView to update/refresh UI.

XMLTreeView()

Description
The class constructor. Creates an object of type `XMLTreeView.`

Syntax
```
public  XMLTreeView();
```

getPreferredSize()

Description
Returns the Dimension object containing the XMLTreeView preferred size. Overrides `getPreferredSize()` in `javax.swing.JComponent` class.

Syntax
```
public java.awt.Dimension getPreferredSize();
```

getTree()

Description

Returns the visual tree as a JTree.

Syntax

```
protected javax.swing.JTree getTree();
```

getXMLTreeModel()

Description

Returns the datamodel for JTree as XMLTreeModel.

Syntax

```
protected oracle.xml.treeviewer.XMLTreeModel getXMLTreeModel();
```

setXMLDocument()

Description

Associates the XMLTreeViewer with a XML document.

Syntax

```
public void setXMLDocument( org.w3c.dom.Document document);
```

Parameter	Description
doc	The Document to display.

updateUI()

Description

Forces the XMLTreeView to update/refresh UI.

Syntax

```
public void updateUI();
```

XMLTreeViewBeanInfo Class

Description of XMLTreeViewBeanInfo

This class provides information about the XMLTreeView Bean.

Syntax of XMLTreeViewBeanInfo

```
public class XMLTreeViewBeanInfo extends java.beans.SimpleBeanInfo

java.lang.Object
  |
  +--java.beans.SimpleBeanInfo
        |
        +--oracle.xml.treeviewer.XMLTreeViewBeanInfo
```

Implemented Interfaces of XMLTreeViewBeanInfo

`java.beans.BeanInfo`

Methods of XMLTreeViewBeanInfo

Table 10–34 Summary of Methods of XMLTreeViewBeanInfo

Method	Description
XMLTreeViewBeanInfo() on page 10-111	Class constructor.
getIcon() on page 10-112	Retrieves an image object representing the requested icon type for XMLTreeView bean in toolbars, toolboxes, and so on.
getPropertyDescriptors() on page 10-112	Retrieves an array of XMLTreeViewl bean's editable PropertyDescriptors.

XMLTreeViewBeanInfo()

Description

Class constructor.

Syntax

```
public  XMLTreeViewBeanInfo();
```

getIcon()

Description

Retrieves an image object representing the requested icon type for XMLTreeView bean in toolbars, toolboxes, and so on.

Syntax

```
public java.awt.Image getIcon( int iconKind);
```

Parameter	Description
iconKind	The kind of icon requested.

getPropertyDescriptors()

Description

Retrieves an array of XMLTreeViewl bean's editable PropertyDescriptors.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

oracle.xml.differ Package

Description

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 10–35.

Table 10–35 Summary of Classes of oracle.xml.differ

Class	Description
XMLDiff Class on page 10-114	Defines an interface for comparing two XML files.
XMLDiffBeanInfo Class on page 10-124	This class provides information about the XMLDiff Bean.

XMLDiff Class

Description of XMLDiff

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 of XMLDiff

```
oracle.xml.differ

java.lang.Object
  |
  +--oracle.xml.differ.XMLDiff
```

Implemented Interfaces of XMLDiff

- `DOMBuilderListener Interface`
- `DOMBuilderErrorListener Interface`
- `java.io.Serializablen Interface`

Methods of XMLDiff

Table 10–36 Summer of Methods of XMLDiff

Method	Description
XMLDiff() on page 10-116	Class constructor.
setFiles() on page 10-116	Sets the XML files which need to be compared.
setDocuments() on page 10-117	Sets the XML documents which need to be compared.
setInput1() on page 10-117	Sets the first XML file or document which need to be compared.
setInput2() on page 10-118	Sets the second XML file or document which need to be compared.
getDocument1() on page 10-118	Gets the document root as an XMLDocument object of the first XML tree.

Table 10–36 Summer of Methods of XMLDiff (Cont.)

Method	Description
getDocument2() on page 10-118	Gets the document root as an XMLDocument object of the second XML tree.
diff() on page 10-119	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 10-119	Retrieves the visual text panel as JTextPane object which visually shows the differences in the first XML file or document.
getDiffPane2() on page 10-119	Retrieves the visual text panel as JTextPane object which visually shows the differences in the second XML file or document.
setIndentIncr() on page 10-119	Sets the indentation for the XSL generation.
setNewNodeIndentIncr() on page 10-120	Generates an XSL file, with user-defined filename, which represents the differences between the two initially set XML files.
generateXSLFile() on page 10-120	Generates an XSL file, with user-defined filename, which represents the differences between the two initially set XML files.
generateXSLDOC() on page 10-121	Generates an XSL stylesheet as an XMLDocument which represents the differences between the two initially set XML documents.
equals() on page 10-121	Compares two nodes. It is called by the differ algorithm. If needed, this function can be overwritten for customized comparisons.
domBuilderErrorCalled() on page 10-121	Method implementing the DOMBuilderErrorListener interface called only by the DOM parser when there is an error while parsing.
domBuilderError() on page 10-122	Method implementing the DOMBuilderErrorListener interface called only by the DOM parser.
domBuilderOver() on page 10-122	Method implementing DOMBuilderListener interface called only by a DOM parser thread when parsing completes.
domBuilderStarted() on page 10-123	Method implementing DOMBuilderListener interface called only by the DOM parser when the parsing starts.

Table 10–36 Summer of Methods of XMLDiff (Cont.)

Method	Description
printDiffTree() on page 10-123	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 10-123	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()

Description

Class constructor.

Syntax

```
public XMLDiff();
```

setFiles()

Description

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()

Description

Sets the XML documents which need to be compared.

Syntax

```
public void setDocuments( XMLDocument doc1,
                          XMLDocument doc2);
```

Parameter	Description
doc1	First XML document.

setInput1()

Description

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()

Description

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()

Description

Gets the document root as an XMLDocument object of the first XML tree.

Syntax

```
public XMLDocument getDocument1();
```

getDocument2()

Description

Gets the document root as an XMLDocument object of the second XML tree.

Syntax

```
public XMLDocument getDocument2();
```

diff()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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.IOException` 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()

Description

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

Description of XMLDiffBeanInfo

This class provides information about the XMLDiff Bean.

Syntax of XMLDiffBeanInfo

```
public class XMLDiffBeanInfo extends java.beans.SimpleBeanInfo

XMLSjava.lang.Object
  |
  +--java.beans.SimpleBeanInfo
        |
        +--oracle.xml.differ.XMLDiffBeanInfo
```

Methods of XMLDiffBeanInfo

Table 10–37 Summary of Methods of XMLDiffBeanInfo

Method	Description
XMLDiffBeanInfo() on page 10-124	Class constructor.
getPropertyDescriptors() on page 10-124	Retrieves an image object representing the requested icon type for XMLDiff bean in toolbars, toolboxes, and so on.
getIcon() on page 10-125	Retrieves an array of XMLDiff bean's editable PropertyDescriptors

XMLDiffBeanInfo()

Description

Class constructor.

Syntax

`public XMLDiffBeanInfo();`

getPropertyDescriptors()

Description

Retrieves an array of XMLDiff bean's editable PropertyDescriptors. Overrides `getPropertyDescriptors()` in `java.beans.SimpleBeanInfo` Class.

Syntax

```
public java.beans.PropertyDescriptor[] getPropertyDescriptors();
```

getIcon()

Description

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 Simple Object Access Protocol (SOAP) for Java

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 and classes:

- 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/`
- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

oracle.soap.server Package

Description of oracle.soap.server

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 11–1 lists the interfaces and classes that provide support for Oracle SOAP in the XDK for Java.

Table 11–1 Summary of Classes and Interfaces of oracle.soap.server

Class/Interface	Description
Handler Interface on page 11-3	Defines the interface for a pluggable handler in the SOAP server.
Provider Interface on page 11-8	Defines the capabilities that must be supported for each type of service provider.
ProviderManager Interface on page 11-11	Defines the Provider Manager used by the SOAP engine to deploy providers, undeploy providers, and access provider deployment information.
ServiceManager Interface on page 11-15	Defines the Service Manager used by the SOAP engine to deploy services, undeploy services, and to access service deployment information.
ContainerContext Class on page 11-19	Defines the context of the container in which the SOAP server is running
Class Logger on page 11-24	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 11-30	Defines the deployment information for a specific provider.
RequestContext Class on page 11-35	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 11-43	Defines the deployment information for a SOAP service, independent of its provider type.
UserContext Class on page 11-59	Defines the user context for a SOAP service request.

Handler Interface

Description of Handler

`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
- be thread-safe

Syntax of Handler

```
oracle.soap.server.Handler

public interface Handler
```

Fields of Handler

Table 11–2 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.

Methods of Handler

Table 11–3 Summary of Methods of Handler

Method	Description
destroy() on page 11-4	Cleans-up handler (one time only).
getName() on page 11-4	Returns this handler's name.
getOptions() on page 11-4	Returns options that are specific to the handler implementation.
init() on page 11-5	Initializes handler (one-time only).

Table 11–3 Summary of Methods of Handler (Cont.)

Method	Description
invoke() on page 11-5	Invokes the requested handler as part of the specified chain type.
setName() on page 11-6	Sets the name of the handler. This method must be called before init().
setOptions() on page 11-6	Sets the options for the handler for subsequent use by init.

destroy()

Description

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()

Description

Returns this handler's name.

Syntax

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

getOptions()

Description

Returns options that are specific to the handler implementation.

Syntax

```
public abstract Properties getOptions();
```

init()

Description

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()

Description

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()

Description

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()

Description

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

Description of Provider

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
- be thread-safe

Syntax of Provider

```
public interface Provider

Interface oracle.soap.server.Provider
```

Methods of Provider

Table 11–4 Summary of Methods in Provider

Method	Description
destroy() on page 11-9	Cleans up provider instance (one time only).
getId() on page 11-9	Returns this providers name, which is unique within the SOAP handler.
init() on page 11-9	Initializes provider instance (one time only).

Table 11–4 Summary of Methods in Provider (Cont.)

Method	Description
invoke() on page 11-10	Invokes the requested method in the specified service, where the SOAP request is completely described in the request context.

destroy()

Description

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()

Description

Returns this providers name, which is unique within the SOAP handler.

Syntax

```
public abstract String getId();
```

init()

Description

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, which contains the logger for informational messages.

invoke()

Description

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	The RequestContext that contains everything the provider needs to process the request.

ProviderManager Interface

Description of ProviderManager

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 of ProviderManager

```
public interface ProviderManager

Interface oracle.soap.server.ProviderManager
```

Methods of ProviderManager

Table 11–5 Summary of Methods in ProviderManager

Method	Description
deploy() on page 11-12	Deploys the given provider.
destroy() on page 11-12	Cleans up the provider manager.
getRequiredRequestURI() on page 11-12	Returns the URI that provider manager requests.
init() on page 11-12	Initializes the provider manager.
list() on page 11-13	Returns an array of provider ids for all providers that have been deployed.
query() on page 11-13	Returns the deployment descriptor for the given provider.
setServiceManager() on page 11-13	Makes the service manager available to the provider manager.
undeploy() on page 11-14	Undeploys the given provider.

deploy()

Description

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()

Description

Cleans up the provider manager. Throws `SOAPException` if unable to cleanup the provider manager.

Syntax

```
public abstract void destroy();
```

getRequiredRequestURI()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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

Description of ServiceManager

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 of ServiceManager

```
public interface ServiceManager

Interface oracle.soap.server.ServiceManager
```

Methods of Servicemanager

Table 11–6 Summary of Methods in ServiceManager

Method	Description
getRequiredRequestURI() on page 11-16	Returns the URI that service manager requests.
deploy() on page 11-16	Deploys the given service.
destroy() on page 11-16	Cleans up the service manager.
init() on page 11-17	Initializes the service manager.
list() on page 11-17	Returns an array of service ids for all services that have been deployed, regardless of the provider.
query() on page 11-17	Returns the deployment descriptor for the given service.
undeploy() on page 11-18	Undeploys the given service, and returns its descriptor.

getRequiredRequestURI()

Description

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()

Description

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()

Description

Cleans up the service manager. Throws `SOAPException` if unable to cleanup the service manager.

Syntax

```
public abstract void destroy();
```

init()

Description

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 the 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()

Description

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()

Description

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()

Description

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

Description of ContainerContext

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 of ContainerContext

```
public class ContainerContext extends Object

java.lang.Object
   |
   +----oracle.soap.server.ContainerContext
```

Fields of ContainerContext

Table 11–7 Fields of ContainerContext

Field	Syntax	Description
SERVLET_CONTAINER	public static final String SERVLET_CONTAINER	The value for a servlet container type.

Methods of ContainerContext

Table 11–8 Summary of Methods of ContainerContext

Method	Description
ContainerContext() on page 11-20	Class constructor.
getAttribute() on page 11-20	Returns the attribute with the given name.
getAttributeNames() on page 11-21	Returns an Enumeration containing the attribute names available within this SOAP context.
getContainerType() on page 11-21	Returns the container type in which the SOAP server is running.
getHttpServlet() on page 11-21	Returns the HTTP servlet that is processing the SOAP request if the container type is SERVLET_CONTAINER.

Table 11–8 Summary of Methods of ContainerContext (Cont.)

Method	Description
removeAttribute() on page 11-21	Removes the attribute with the given name from the context.
setAttribute() on page 11-22	Binds an object to a given attribute name in this SOAP context.
setContainerType() on page 11-22	Sets the container type.
setHttpServlet() on page 11-23	Sets the HTTP servlet for a SOAP server running in a SERVLET_CONTAINER type of container.

ContainerContext()

Description

Class constructor.

Syntax

```
public ContainerContext();
```

getAttribute()

Description

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()

Description

Returns an `Enumeration` containing the attribute names available within this SOAP context.

Syntax

```
public Enumeration getAttributeNames();
```

getContainerType()

Description

Returns the container type in which the SOAP server is running.

Syntax

```
public String getContainerType();
```

getHttpServlet()

Description

Returns the HTTP servlet that is processing the SOAP request if the container type is `SERVLET_CONTAINER`, or `NULL` if the servlet attribute is not set.

Syntax

```
public HttpServlet getHttpServlet();
```

removeAttribute()

Description

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()

Description

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()

Description

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()

Description

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

Description of 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 of Logger

```
Class oracle.soap.server.Logger

public abstract class Logger extends Object

java.lang.Object
   |
   +----oracle.soap.server.Logger
```

Fields of Logger

Table 11–9 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.

Table 11–9 Fields of Logger (Cont.)

Field	Syntax	Description
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.
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 11–10 Summary of Methods of Logger

Method	Description
Logger() on page 11-26	Class constructor.
getSeverity() on page 11-26	Returns the current severity setting for the logger.
getSeverityName() on page 11-26	Returns the severity name associated with the given severity.
getSeverityValue() on page 11-26	Returns the severity value associated with the given severity name.
init() on page 11-27	Initializes of the logger (one-time only) with its configuration parameters.
isLoggable() on page 11-27	Determines if a message would be logged at the given severity level.
log() on page 11-28	Logs messages.
setSeverity() on page 11-28	Sets the current severity.

Logger()

Description
Class constructor.

Syntax
```
public Logger();
```

getSeverity()

Description
Returns the current severity setting for the logger.

Syntax
```
public int getSeverity();
```

getSeverityName()

Description
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()

Description
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()

Description

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()

Description

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()

Description

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()

Description

Sets the current severity.

Syntax

```
public void setSeverity(int severity);
```

Parameter	Description
severity	The new severity setting for the logger.

ProviderDeploymentDescriptor Class

Description of ProviderDeploymentDescriptor

`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.

Syntax of ProviderDeploymentDescriptor

```
public final class ProviderDeploymentDescriptor extends Object implements
Serializable

java.lang.Object
   |
   +----oracle.soap.server.ProviderDeploymentDescriptor
```

Methods of ProviderDeploymentDescriptor

Table 11–11 Summary of Methods of ProviderDeploymentDescriptor

Method	Descriptor
ProviderDeploymentDescriptor() on page 11-31	Constructs a new instance of a provider descriptor.
fromXML() on page 11-31	Builds and returns a provider descriptor from the given XML document.
getClassname() on page 11-31	Returns the name of the class that implements this provider.
getId() on page 11-31	Returns the unique id for this provider.
getOptions() on page 11-32	Returns the provider-specific options
getProviderType() on page 11-32	Returns the provider type.
setClassname() on page 11-32	Sets the name of the class that implements this provider.
setId() on page 11-32	Sets the provider id.
setOptions() on page 11-33	Sets the options.
setProviderType() on page 11-33	Sets the provider type.
toString() on page 11-33	Writes out the service deployment descriptor to String.
toXML() on page 11-34	Writes out the service deployment descriptor as XML.

ProviderDeploymentDescriptor()

Description

Constructs a new instance of a provider descriptor.

Syntax

```
public ProviderDeploymentDescriptor();
```

fromXML()

Description

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()

Description

Returns the name of the class that implements this provider.

Syntax

```
public String getClassname();
```

getId()

Description

Returns the unique id for this provider.

Syntax

```
public String getId();
```

getOptions()

Description

Returns the provider-specific options, or value pairs that represent the provider-specific options for this service.

Syntax

```
public Hashtable getOptions();
```

getProviderType()

Description

Returns the provider type.

Syntax

```
public String getProviderType();
```

setClassname()

Description

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()

Description

Sets the provider id.

Syntax

```
public void setId( String id);
```

Parameter	Description
id	The unique provider id.

setOptions()

Description

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()

Description

Sets the provider type.

Syntax

```
public void setProviderType( String providerType);
```

Parameter	Description
providerType	The provider type.

toString()

Description

Writes out the service deployment descriptor to String.

Syntax

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

toXML()

Description

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

Description of RequestContext

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. Note that 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 of RequestContext

```
public class RequestContext extends Object

java.lang.Object
   |
   +----oracle.soap.server.RequestContext
```

Methods of RequestContext

Table 11–12 Summary of Methods of RequestContext

Method	Description
RequestContext() on page 11-37	Default constructor for this class.
getMethodName() on page 11-37	Returns the method name being invoked for this SOAP request.
getRequestEncodingStyle() on page 11-37	Returns the encoding style that was used on the request.
getRequestEnvelope() on page 11-38	Returns the envelope that represents the actual SOAP request.
getResponseBytes() on page 11-38	Returns the response stream for this SOAP request.
getResponseEnvelope() on page 11-38	Returns the envelope that represents the SOAP response.
getResponseMap() on page 11-38	Returns the mapping registry that must be used to serialize the SOAP response.
getServiceDeploymentDescriptor() on page 11-39	Returns the service deployment descriptor for the requested service.
getServiceId() on page 11-39	Returns the service id (URI) for this SOAP request.
getUserContext() on page 11-39	Returns the user context for this SOAP request.
setMethodName() on page 11-39	Sets the method name for this SOAP request.
setRequestEncodingStyle() on page 11-40	Sets the encoding style that was used on the request.
setRequestEnvelope() on page 11-40	Sets the envelope that represents the actual SOAP request.
setResponseBytes() on page 11-40	Sets the response stream for this SOAP request.
setResponseEnvelope() on page 11-41	Sets the envelope that represents the SOAP response.

Table 11–12 Summary of Methods of RequestContext

Method	Description
setResponseMap() on page 11-41	Sets the mapping registry that must be used to serialize the SOAP response envelope.
setServiceDeploymentDescriptor on page 11-41	Sets the service deployment descriptor for the requested service.
setServiceId() on page 11-42	Sets the service id (URI) for this SOAP request.
setUserContext() on page 11-42	Sets the user context for this SOAP request.

RequestContext()

Description

Default constructor for this class.

Syntax

```
public RequestContext();
```

getMethodName()

Description

Returns the method name being invoked for this SOAP request.

Syntax

```
public String getMethodName();
```

getRequestEncodingStyle()

Description

Returns the encoding style that was used on the request.

Syntax

```
public String getRequestEncodingStyle();
```

getRequestEnvelope()

Description

Returns the envelope that represents the actual SOAP request.

Syntax

```
public Envelope getRequestEnvelope();
```

getResponseBytes()

Description

Returns the response stream for this SOAP request.

Syntax

```
public ByteArrayOutputStream getResponseBytes();
```

getResponseEnvelope()

Description

Returns the envelope that represents the SOAP response.

Syntax

```
public Envelope getResponseEnvelope();
```

Parameter	Description
smr	The mapping registry for the SOAP response envelope.

getResponseMap()

Description

Returns the mapping registry that must be used to serialize the SOAP response.

Syntax

```
public SOAPMappingRegistry getResponseMap();
```

getServiceDeploymentDescriptor()

Description

Returns the service deployment descriptor for the requested service, or `NULL` if the provider is an AutonomousProvider.

Syntax

```
public ServiceDeploymentDescriptor getServiceDeploymentDescriptor();
```

getServiceId()

Description

Returns the service id (URI) for this SOAP request.

Syntax

```
public String getServiceId();
```

getUserContext()

Description

Returns the user context for this SOAP request.

Syntax

```
public UserContext getUserContext();
```

setMethodName()

Description

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()

Description

Sets the encoding style that was used on the request.

Syntax

```
public void setRequestEncodingStyle( String requestEncodingStyle);
```

Parameter	Description
requestEncodingStyle	The request encoding style.

setRequestEnvelope()

Description

Sets the envelope that represents the actual SOAP request.

Syntax

```
public void setRequestEnvelope( Envelope envelope);
```

Parameter	Description
envelope	The SOAP envelope.

setResponseBytes()

Description

Sets the response stream for this SOAP request.

Syntax

```
public void setResponseBytes( ByteArrayOutputStream bytes);
```

Parameter	Description
bytes	The ByteArrayOutputStream that contains the response.

setResponseEnvelope()

Description

Sets the envelope that represents the SOAP response.

Syntax

```
public void setResponseEnvelope( Envelope envelope);
```

Parameter	Description
envelope	The SOAP response envelope.

setResponseMap()

Description

Sets the mapping registry that must be used to serialize the SOAP response envelope.

Syntax

```
public void setResponseMap( SOAPMappingRegistry smr);
```

setServiceDeploymentDescriptor

Description

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()

Description

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()

Description

Sets the user context for this SOAP request.

Syntax

```
public void setUserContext( UserContext userContext);
```

Parameter	Description
userContext	The user context.

ServiceDeploymentDescriptor Class

Description of ServiceDeploymentDescriptor

`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 of ServiceDeploymentDescriptor

```
public final class ServiceDeploymentDescriptor extends Object implements
Serializable

java.lang.Object
   |
   +----oracle.soap.server.ServiceDeploymentDescriptor
```

Fields of ServiceDeploymentDescriptor

Table 11–13 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.

Methods of ServiceDeploymentDescriptor

Table 11–14 Summary of Methods of ServiceDeploymentDescriptor

Method	Description
ServiceDeploymentDescriptor() on page 11-45	Constructs a new service descriptor.
buildFaultRouter() on page 11-45	Returns the fault router that is built from the service's fault listeners.
buildSOAPMappingRegistry() on page 11-46	Generates an XML serialization registry from all the type mappings registered into a deployment descriptor.
buildSqlClassMap() on page 11-46	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 11-46	Populates the ServiceDeploymentDescriptor with information from the given document, which is the XML representation of the descriptor.
getDefaultSMRClass() on page 11-47	Returns the default SOAP mapping registry class.
getFaultListener() on page 11-47	Returns list of class names that are fault listeners for this service.
getId() on page 11-47	Returns the service id, which is a URI.
getMethods() on page 11-47	Returns the list of methods that are provided by this service.
getProviderId() on page 11-48	Returns the provider id for this service.
getProviderOptions() on page 11-48	Returns the name-value pairs that represent the provider-specific options for this service.
getProviderType() on page 11-48	Returns the provider type.
getScope() on page 11-48	Returns the scope, which is one of the SCOPE_xxx constants.
getServiceType() on page 49	Returns the service type, which is one of the SERVICE_TYPE_ xxx constants.
getSqlMap() on page 11-49	Returns the SQL type to Java type map.
getTypeMappings() on page 49	Returns the XML-Java type mappings, which define how to deserialize XML into Java and serialize Java into XML.
isMethodValid() on page 11-49	Determines if the given method is valid for this service.
setDefaultSMRClass() on page 11-50	Sets the default SOAP mapping registry class.

Table 11–14 Summary of Methods of ServiceDeploymentDescriptor (Cont.)

Method	Description
setFaultListener() on page 11-50	Sets the fault listener list.
setId() on page 11-50	Sets the service id, which must be a valid URI.
setMethods() on page 11-51	Sets the list of methods that are provided by this service.
setProviderId() on page 11-51	Sets the id of the provider for this service.
setProviderOptions() on page 11-51	Sets the provider-specific options.
setProviderType() on page 11-52	Sets the provider type.
setScope() on page 11-52	Sets the execution scope.
setServiceType() on page 11-52	Sets the service type.
setSqlMap() on page 11-53	Sets the map that maps from SQL type to Java type.
setTypeMappings() on page 11-53	Sets the XML-Java type mappings, which define how to deserialize XML into Java and serialize Java into XML.
toXML() on page 11-54	Writes out the service deployment descriptor as XML.
toString() on page 11-54	Returns a printable representation of this descriptor.

ServiceDeploymentDescriptor()

Description

Constructs a new service descriptor.

Syntax

```
public ServiceDeploymentDescriptor();
```

buildFaultRouter()

Description

Returns the fault router that is built from the service's fault listeners.

Syntax

```
public SOAPFaultRouter buildFaultRouter();
```

buildSOAPMappingRegistry()

Description

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()

Description

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()

Description

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()

Description

Returns the default SOAP mapping registry class.

Syntax

```
public String getDefaultSMRClass();
```

getFaultListener()

Description

Returns list of class names that are fault listeners for this service.

Syntax

```
public String[] getFaultListener();
```

getId()

Description

Returns the service id, which is a URI.

Syntax

```
public String getId();
```

getMethods()

Description

Returns the list of methods that are provided by this service.

Syntax

```
public String[] getMethods();
```

getProviderId()

Description

Returns the provider id for this service.

Syntax

```
public String getProviderId();
```

getProviderOptions()

Description

Returns the name-value pairs that represent the provider-specific options for this service.

Syntax

```
public Hashtable getProviderOptions();
```

getProviderType()

Description

Returns the provider type.

Syntax

```
public String getProviderType();
```

getScope()

Description

Returns the scope, which is one of the `SCOPE_xxx` constants.

Syntax

```
public int getScope();
```

getServiceType()

Description

Returns the service type, which is one of the SERVICE_TYPE_xxx constants.

Syntax

```
public int getServiceType();
```

getSqlMap()

Description

Returns the SQL type to Java type map.

Syntax

```
public Hashtable getSqlMap();
```

getTypeMappings()

Description

Returns the XML-Java type mappings, which define how to deserialize XML into Java and serialize Java into XML.

Syntax

```
public TypeMapping[] getTypeMappings();
```

isMethodValid()

Description

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()

Description

Sets the default SOAP mapping registry class.

Syntax

```
public void setDefaultSMRClass( String defaultSMRClass);
```

Parameter	Description
defaultSMRClass	The default SOAP mapping registry class.

setFaultListener()

Description

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()

Description

Sets the service id, which must be a valid URI.

Syntax

```
public void setId( String id);
```

Parameter	Description
id	The service URI.

setMethods()

Description

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()

Description

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()

Description

Sets the provider-specific options.

Syntax

```
public void setProviderOptions( Hashtable providerOptions);
```

Parameter	Description
providerOptions	The name-value pairs that represent the provider-specific options for this service.

setProviderType()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Writes out the service deployment descriptor as XML.

Syntax

```
public void toXML( Writer pr);
```

Parameter	Description
pr	The writer for the XML output.

toString()

Description

Returns a printable representation of this descriptor.

Syntax

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

SOAPServerContext Class

Description of SOAPServerContext

SOAPServerContext defines the context of the SOAP server that is independent of the type of container in which the server is running.

Syntax of SOAPServerContext

```
public class SOAPServerContext extends Object

java.lang.Object
   |
   +----oracle.soap.server.SOAPServerContext
```

Methods of SOAPServerContext

Table 11–15 Summary of Methods of SOAPServerContext

Method	Description
SOAPServerContext() on page 11-56	Default constructor.
getAttribute() on page 11-56	Returns the attribute with the given name, or null if there is no attribute by that name.
getAttributeNames() on page 11-56	Returns an Enumeration containing the attribute names available within this SOAP context.
getGlobalContext() on page 11-56	Returns the global context.
getLogger() on page 11-57	Returns the SOAP logger.
removeAttribute() on page 11-57	Removes the attribute with the given name from the context.
setAttribute() on page 11-57	Binds an object to a given attribute name in this SOAP context.
setGlobalContext () on page 11-58	Set the global context, which contains SOAP server-wide objects.
setLogger() on page 11-58	Set the logger, which is used for text-based logging of informational and debug messages.

SOAPServerContext()

Description

Default constructor.

Syntax

```
public SOAPServerContext();
```

getAttribute()

Description

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()

Description

Returns an Enumeration containing the attribute names available within this SOAP context.

Syntax

```
public Enumeration getAttributeNames();
```

getGlobalContext()

Description

Returns the global context that contains SOAP server-wide objects, or null if the attribute is not set.

Syntax

```
public Hashtable getGlobalContext();
```

getLogger()

Description

Returns the SOAP logger, which is used to log informational and debug messages.

Syntax

```
public Logger getLogger();
```

removeAttribute()

Description

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()

Description

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 ()

Description

Sets the global context, which contains SOAP server-wide objects.

Syntax

```
public void setGlobalContext( Hashtable globalContext);
```

Parameter	Description
globalContext	The global context.

setLogger()

Description

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

Description of UserContext

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.

Note that the `HttpServlet` and `HttpSession` do not really belong here, but they are required by the JavaProvider.

Syntax of UserContext

```
public class UserContext extends Object

java.lang.Object
   |
   +----oracle.soap.server.UserContext
```

Methods of UserContext

Table 11–16 Summary of Methods of UserContext

Method	Description
UserContext() on page 11-60	Default constructor.
getAttribute() on page 11-60	Returns the attribute with the given name
getAttributeNames() on page 11-61	Returns an Enumeration containing the attribute names available within this SOAP context.
getCertificate() on page 61	Returns the user certificate for the user making SOAP request.
getHttpServlet() on page 11-61	Returns the HttpServlet that is processing the SOAP request.
getHttpSession() on page 11-61	Returns the HTTP session for the SOAP request
getRemoteAddress() on page 11-62	Returns the Internet Protocol (IP) address of the remote client that sent the request.
getRemoteHost() on page 11-62	Returns the host name of the remote client that sent the request.
getRequestURI() on page 11-62	Returns the URI of the request.
getSecureChannel() on page 62	Returns an indication whether the channel is secure.
getUsername() on page 63	Returns the protocol-specific username for the SOAP request.

Table 11–16 Summary of Methods of UserContext (Cont.)

Method	Description
removeAttribute() on page 11-63	Removes the attribute with the given name from the context.
setAttribute() on page 11-63	Binds an object to a given attribute name in this SOAP context.
setCertificate() on page 11-64	Sets the user certificate.
setHttpServlet() on page 11-64	Sets the HTTP servlet.
setHttpSession() on page 64	Sets the HTTP session.
setRemoteAddress() on page 11-65	Sets the remote IP address of the client.
setRemoteHost() on page 11-65	Sets the host name of the remote client making the SOAP request.
setRequestURI() on page 11-65	Sets the URI of the request.
setSecureChannel() on page 11-66	Sets the indicator of whether the channel is secure.
setUsername() on page 11-66	Sets the protocol-specific username.

UserContext()

Description

Default constructor.

Syntax

```
public UserContext();
```

getAttribute()

Description

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()

Description

Returns an `Enumeration` containing the attribute names available within this SOAP context.

Syntax

```
public Enumeration getAttributeNames();
```

getCertificate()

Description

Returns the user certificate for the user making SOAP request, or null if this attribute is not set.

Syntax

```
public Object getCertificate();
```

getHttpServlet()

Description

Returns the HttpServlet that is processing the SOAP request, or null if the servlet attribute is not set.

Syntax

```
public HttpServlet getHttpServlet();
```

getHttpSession()

Description

Returns the HTTP session for the SOAP request, or null if the session attribute is not set.

Syntax

```
getHttpSession public HttpSession getHttpSession();
```

getRemoteAddress()

Description

Returns the Internet Protocol (IP) address of the remote client that sent the request.

Syntax

```
public String getRemoteAddress();
```

getRemoteHost()

Description

Returns the host name of the remote client that sent the request.

Syntax

```
public String getRemoteHost();
```

getRequestURI()

Description

Returns the URI of the request.

Syntax

```
public String getRequestURI();
```

getSecureChannel()

Description

Returns an indication whether the channel is secure; `TRUE` if the channel is secure, `FALSE` otherwise.

Syntax

```
public boolean getSecureChannel();
```

getUsername()

Description

Returns the protocol-specific username for the SOAP request, or null if this attribute is not set.

Syntax

```
public String getUsername();
```

removeAttribute()

Description

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()

Description

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.

setCertificate()

Description
Sets the user certificate.

Syntax
```
public void setCertificate( Object certificate);
```

Parameter	Description
certificate	The user certificate for the user making the SOAP request.

setHttpServlet()

Description
Sets the HTTP servlet.

Syntax
```
public void setHttpServlet( HttpServlet servlet);
```

Parameter	Description
servlet	The HttpServlet that is processing the SOAP request.

setHttpSession()

Description
Sets the HTTP session.

Syntax

```
public void setHttpSession( HttpSession session);
```

Parameter	Description
servlet	The HttpSession for the SOAP request.

setRemoteAddress()

Description

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()

Description

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()

Description

Sets the URI of the request.

Syntax

```
public void setRequestURI( String uri);
```

Parameter	Description
uri	Request URI.

setSecureChannel()

Description

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()

Description

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

Description of oracle.soap.transport

This package contains the OracleSOAPTransport Interface, which provides support for Oracle SOAP in the XDK for Java.

OracleSOAPTransport Interface

Description of OracleSOAPTransport

This interface defines Oracle specific transport extensions.

Syntax of OracleSOAPTransport

`oracle.soap.transport.OracleSOAPTransport`

```
public interface OracleSOAPTransport extends SOAPTransport
```

Methods of OracleSOAPTransport

Table 11–17 Summary of Methods of OracleSOAPTransport

Method	Description
close() on page 11-67	Close the transport and perform any clean up.
getProperties() on page 11-68	Returns the connection properties.
setProperties() on page 11-68	Sets the connection properties.

close()

Description

Closes the transport and performs clean up.

Syntax

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

getProperties()

Description

Returns the connection properties.

Syntax

```
public abstract Properties getProperties();
```

setProperties()

Description

Sets the connection properties.

Syntax

```
public abstract void setProperties( Properties prop);
```

Parameter	Description
prop	Connection properties.

oracle.soap.transport.http Package

Description of oracle.soap.transport.http

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

Description of OracleSOAPHTTPConnection

This class implements `OracleSOAPTransport`.

Syntax of OracleSOAPHTTPConnection

```
public class OracleSOAPHTTPConnection extends Object

java.lang.Object
   |
   +----oracle.soap.transport.http.OracleSOAPHTTPConnection
```

Fields of OracleSOAPHTTPConnection

Table 11–18 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.

Table 11–18 Fields of OracleSOAPHTTPConnection (Cont.)

Field	Syntax	Description
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.
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.

Methods of OracleSOAPHTTPConnection

Table 11–19 Summary of Methods of OracleSOAPHTTPConnection

Member	Description
OracleSOAPHTTPConnection() on page 11-71	Constructs a new instance of `OracleSOAPHTTPConnection` from given properties.
close() on page 11-71	Closes the connection.
finalize() on page 11-72	Finalizes the connection.
getHeaders() on page 11-72	Returns a hashtable containing all the headers to headers generated by the protocol.

Table 11–19 Summary of Methods of OracleSOAPHTTPConnection (Cont.)

Member	Description
getProperties() on page 11-72	Returns the connection properties.
receive() on page 72	Returns a buffered reader from which the received response is read.
send() on page 11-73	Requests that an envelope be posted to the given URL.
setProperties() on page 11-73	Sets the connection properties.

OracleSOAPHTTPConnection()

Description

Constructs a new instance of `OracleSOAPHTTPConnection` from given properties.

Syntax

```
public OracleSOAPHTTPConnection( Properties prop);
```

Parameter	Description
prop	Connection properties.

close()

Description

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()

Description

Finalizes the connection.

Syntax

```
public void finalize();
```

getHeaders()

Description

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()

Description

Returns the connection properties.

Syntax

```
public Properties getProperties();
```

receive()

Description

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()

Description

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()

Description

Sets the connection properties.

Syntax

```
public void setProperties( Properties prop);
```

Parameter	Description
prop	Connection properties.

oracle.soap.util.xml Package

Description of oracle.soap.util.xml

Package oracle.soap.util.xml contains the XmlUtils Class.

XmlUtils Class

Description of XmlUtils

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 of XmlUtils

```
public class XmlUtils

java.lang.Object
   |
   +----oracle.soap.util.xml.XmlUtils
```

Methods of XmlUtils

Table 11–20   Summary of Methods of XmlUtils

Member	Description
XmlUtils() on page 11-75	Default constructor.
extractServiceId() on page 11-75	Returns the service id from the envelope.
extractMethodName() on page 11-75	Returns the method name from the envelope.
parseXml() on page 11-76	Parses the given XML file and returns the XML document.
createDocument() on page 11-76	Creates a Document.

XmlUtils()

Description

Default constructor.

Syntax

```
public XmlUtils();
```

extractServiceId()

Description

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()

Description

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()

Description

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()

Description

Creates a `Document`. Throws a `SOAPException` if cannot create `Document`.

Syntax

```
public static Document createDocument();
```

12

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.

The following pages describe the following:

- TransX Utility Command Line Interface
- TransX Utility Application Programming Interface
  - loader Class
  - TransX Interface

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied Java Packages Reference*

TransX Utility Command Line Interface

TransX Utility Command Line 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)]
```

TransX Utility Command Line Parameters

Table 12–1 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 12–2, "TransX Utility Command Line Options".

TransX Utility Command Line Options and Exceptions

Table 12–2 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.

Table 12–2 TransX Utility Command Line Options (Cont.)

Option	Description
-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.
-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 12–3 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

Description of loader

Provides the methods to instantiate the TransX Interface, through which loading operations are performed.

Syntax of loader

```
oracle.xml.transx.loader
```

Methods of loader

Table 12–4 Summary of Methods of loader

Method	Description
getLoader() on page 12-4	Get an instance of TransX.
main() on page 12-4	The command-line interface.

getLoader()

Description

Get an instance of TransX. See also TransX Interface.

Syntax

```
public static TransX getLoader();
```

main()

Description

The command-line interface main function.

Syntax

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

Parameter	Description
args	Command line parameters.

TransX Interface

Description

Data Loading Tool API.

Syntax

```
public interface TransX
```

Methods

Table 12–5 Summary of Methods of TransX

Method	Description
close() on page 12-6	Help reclaim used resources.
load() on page 12-6	Load a dataset on a file or URL.
open() on page 12-6	Start a data loading session.
setLoadingMode() on page 12-7	Set the operation mode on duplicates.
setPreserveWhitespace() on page 12-8	Tell the loader to treat white spaces as significant.
setValidationMode() on page 12-8	Set the validation mode.
unload() on page 12-9	Unload a dataset.
validate() on page 12-9	Validate a dataset on a file or URL.

close()

Description

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()

Description

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 speficied 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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Validate a dataset on a file or URL. Once the dataset has been 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, `FALSE` otherwise. Reports an `oracle.xml.parser.schema.XSDException.`

Syntax

```
public boolean validate( String datasrc);
```

Parameter	Description
datasrc	A file name or URL string

Part II

XDK for C Packages

This section contains the following chapters:

13 XML Parser for C

This chapter describes the following sections:

- Parser APIs
- W3C SAX APIs
- W3C DOM APIs
- Namespace APIs

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*

Parser APIs

This C implementation of the XML processor (or parser) follows the W3C XML specification (rev REC-xml-19980210) and implements 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.

Calling Sequence

Parsing a single document:

`xmlinit, xmlparse`***`xxx`***`, xmlterm`

Parsing multiple documents, but only the latest document's data needs to be available:

`xmlinit, xmlparse`***`xxx`***`, xmlclean, xmlparse`***`xxx`***`, xmlclean ... xmlterm`

Parsing multiple documents, all document data must be available:

`xmlinit, xmlparse`***`xxx`***`, xmlparse`***`xxx`***` ... xmlterm`

Memory

Memory callback functions may be used if you wish to use your own memory allocation. If they are used, both 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:

- `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.

Data Types

Table 13–1 Summary of Data Types for Parser APIs

Data Type	Syntax	Description
boolean	typedef int boolean;	Boolean value, TRUE or FALSE.
oratext	typedef unsigned char oratext;	String pointer used for all data encodings, cast as needed; for UTF-16, to `(ub2 *)`
string	typedef unsigned char String;	String pointer (C/C++)
ub4	typedef unsigned int ub4;	32-bit (or larger) unsigned integer
uword	typedef unsigned int uword;	Native unsigned integer
xmlacctype	XMLACCESS_UNK /*An access method was specified an a URL, but it was unknown to the XML parser*/ XMLACCESS_FILE /*Filesystem I/O*/ XMLACCESS_HTTP /*HyperText Transport Protocol; the Web)*/ XMLACCESS_FTP /*File Transfer Protocol */ XMLACCESS_GOPHER /*Gopher*/ XMLACCESS_ORADB /*Direct Oracle database access*/ XMLACCESS_STREAM /*User-defined stream*/	An enumeration of the possible access methods used to retrieve an XML document XML access type, like HTTP, FTP, File, and so on. See the `xmlaccess()` function for more details.
xmlctx	typedef struct xmlctx xmlctx;	Top-level XML context; the 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); }; typedef struct xmlmemcb xmlmemcb;	Memory callback structure (optional). This is the memory callback structure. Allocations do not need to be initialized; works like `malloc`, not `calloc`.
xmlnode	typedef struct xmlnode xmlnode;	Note: The contents of xmlnode are private and must not be accessed by users.

Table 13–1 Summary of Data Types for Parser APIs (Cont.)

Data Type	Syntax	Description

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 */
		Node type enumeration
		
		Parse tree node types, see `getNodeType()`. Names and values match DOM specification.

Table 13–1 Summary of Data Types for Parser APIs (Cont.)

Data Type	**Syntax**	**Description**
xmlsaxc	struct xmlsaxcb	SAX callback structure (SAX only)
	{	
	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 (*comment)(void *ctx,	
	const oratext *data);	
	sword (*elementDecl)(void *ctx,	
	const oratext *name,	
	const oratext *content);	
	sword (*attributeDecl)(void *ctx,	
	const oratext *elem,	
	const oratext *attr,	
	const oratext *body);	
	sword (*xmlDecl)(void *ctx,	
	const oratext *version,	
	boolean encoding);	
	/* Following fields are reserved for future use.*/	
	void (*empty1)();	
	void (*empty2)();	
	void (*empty3)();	
	void (*empty4)();	
	};	
	typedef struct xmlsaxcb xmlsaxcb;	

Functions and Methods of Parser APIs

Table 13–2 Summary of Functions and Methods of Parser APIs

Function / Method	Description
freeElements() on page 13-7	Frees an allocated list of element nodes.
getEncoding() on page 13-7	Returns document's encoding.
isSingleChar() on page 13-7	Determines if document data is single or multibyte.
isStandalone() on page 13-8	Determines if document is standalone.
isUnicode() on page 13-8	Determines if document data is Unicode.
saveString() on page 13-9	Allocates memory and saves the NULL-terminated single or multibyte string in the XML string pool.
saveString2() on page 13-9	Allocates memory and saves the NULL-terminated Unicode string in the XML string pool.
printBuffer() on page 13-10	Prints representation of XML tree into buffer.
printSize() on page 13-10	Returns size of printed representation of XML tree.
printStream() on page 13-11	Writes a printed representation of an XML tree to file stream.
setDocOrder() on page 13-11	Sets the document order for each node in the current document.
xmlaccess() on page 13-12	Sets the I/O callback functions for the given access method.
xmlclean() on page 13-14	Frees any memory used during the previous parse.
xmlinit() on page 13-14	Initializes XML parser.
xmlinitenc() on page 13-17	Initializes XML parser specifying DOM data encoding.
xmlLocation() on page 13-18	Returns current location while parsing.
xmlparse() on page 13-19	Parses a URI.
xmlparsebuf() on page 13-20	Parses a buffer.
xmlparsedtd() on page 13-22	Parses an external DTD.
xmlparsefile() on page 13-24	Parses a file.
xmlparsestream() on page 13-25	Parses a user-defined stream.
xmlterm() on page 13-27	Shuts down XML parser.
xmlwhere() on page 13-28	Returns error location information.

freeElements()

Description

Frees an allocated list of element nodes. Used primarily to free the lists created by `getElementsByTagName()`.

Syntax

```
void freeElements( xmlctx *ctx,
                   xmlnodes *list);
```

Parameter	IN / OUT	Description
ctx	(IN)	XML context.
list	(IN)	List of nodes to free

getEncoding()

Description

This function returns the IANA / Mime name of the DOM / SAX data encoding, such as "`ASCII`", "`ISO-8859-1`", "`UTF-8`", "`UTF-16`", and so on. See also the `isSingleChar()` function, which can be used to determine if the data is single or multibyte, and the `isUnicode()` function, which determines if the data is Unicode (`UTF-16`). The data encoding is specified by the user at initiation time.

Syntax

```
oratext *getEncoding( xmlctx *ctx);
```

Parameter	IN / OUT	Description
ctx	(IN)	The XML parser context.

isSingleChar()

Description

Returns a flag which specifies whether data encoding to this context is singlebyte characters (like `ASCII, ISO-8859, EBCDIC,` and others), or multibyte characters (like `UTF-8` or `Unicode`). See `getEncoding()`, which returns the name of the data encoding.

Syntax

```
boolean isSingleChar( xmlctx *ctx);
```

Parameter	IN / OUT	Description
ctx	(IN)	The XML parser context

isStandalone()

Description

Returns value of document's standalone flag. This function returns the boolean value of the document's *standalone* flag, as specified in the XML declaration.

Syntax

```
boolean isStandalone( xmlctx *ctx);
```

Parameter	IN / OUT	Description
ctx	(IN)	The XML parser context

isUnicode()

Description

Returns the Unicode (UCS2) encoding flag. Similar to a `isSingleChar()`.

Syntax

```
boolean isUnicode(xmlctx *ctx);
```

Parameter	IN / OUT	Description
ctx	(IN)	The XML parser context

saveString()

Description

Allocates memory and saves the `NULL`-terminated single or multibyte string in the XML string pool. Strings saved this way cannot be freed individually since they are stored head-to-tail in a single pool, for maximum compactness. The memory is reused only when the entire pool is freed, after an `xmlclean()` or `xmlterm()` calls. Use `saveString2()` for saving Unicode strings.

Syntax

```
oratext *saveString( xmlctx *ctx,
                     oratext *str);
```

Parameter	IN / OUT	Description
xtx	(IN)	LPX context
str	(IN)	Pointer to a single or multibyte string.

saveString2()

Description

Allocates memory and saves the NULL-terminated Unicode string in the XML string pool. Note that a Unicode string is terminated with TWO NULL bytes, not just one! Strings saved this way cannot be freed individually since they are stored head-to-tail in a single pool, for maximum compactness. The memory is reused only when the entire pool is freed, after an `xmlclean()` or `xmlterm()` calls. Use saveString() for saving single or multibyte strings.

Syntax

```
ub2 *saveString2( xmlctx *ctx,
                  ub2 *ustr);
```

Parameter	IN / OUT	Description
xtx	(IN)	LPX context
ustr	(IN)	Pointer to a unicode string

printBuffer()

Description

Creates a printed representation of an XML tree rooted at the given node, and puts it into a destination buffer. Indentation is controlled by level and step: step is the number of spaces to indent each new level, and level is the starting level; 0 for top-level.

Syntax

```
void printBuffer( oratext *buffer,
                  size_t bufsiz,
                  xmlnode *node,
                  uword step,
                  uword level);
```

Parameter	IN / OUT	Description
buffer	(IN)	destination buffer where output is placed
bufsiz	(IN)	size of destination buffer
node	(IN)	root node of XML tree to print
step	(IN)	number of spaces to indent each new level
level	(IN)	starting level of indentation

printSize()

Description

Returns the size of the printed representation of an XML tree, rooted at the given node. Indentation is controlled by level and step as for printBuffer: step is the number of spaces to indent each new level, and level is the starting level; 0 for top-level. This function is used to pre-compute the size of the buffer needed for `printBuffer()`.

Syntax

```
size_t printSize( xmlnode *node,
                  uword step,
                  uword level);
```

Parameter	IN / OUT	Description
node	(IN)	root node of XML tree to print
step	(IN)	number of spaces to indent each new level
level	(IN)	starting level of indentation

printStream()

Description

Writes a printed representation of an XML tree (rooted at the given node) to a stdio stream (FILE*). This function is exactly like printBuffer except output is to a stream instead of into a buffer. Indentation is controlled by level and step: step is the number of spaces to indent each new level, and level is the starting level (0 for top-level).

Syntax

```
void printStream( FILE *stream,
                  xmlnode *node,
                  uword step,
                  uword level);
```

Parameter	IN / OUT	Description
stream	(IN)	output stream to write to
node	(IN)	root node of XML tree to print
step	(IN)	number of spaces to indent each new level
level	(IN)	starting level of indentation

setDocOrder()

Description

Sets the document order for each node in the current document. Must be called once on the final document before XSLT processing can occur. Note this is called automatically by the XSLT processor, so ordinarily the user need not make this call.

Syntax

```
ub4 setDocOrder(xmlctx *ctx, ub4 start_id);
```

Parameter	IN / OUT	Description
ctx	(IN)	XML context
start_id	((N)	Initial id number to assign

xmlaccess()

Description

Sets the I/O callback functions for the given access method.

Syntax

```
uword xmlaccess( xmlctx *ctx,
                 xmlacctype access,
                 XML_OPENF((*openf)),
                 XML_CLOSEF((*closef)),
                 XML_READF((*readf)));
```

Parameter	IN / OUT	Description
ctx	(IN)	The XML context
access	(IN)	Access method enum, XMLACCESS_xxx
openf	(IN)	Open-input callback function
closef	(IN)	Close-input callback function
readf	(IN)	Read-input callback function

Comments

Sets the I/O callback functions for the given access method. Most methods have built-in callback functions, so do not have to be provided by the user. The notable exception is `XMLACCESS_STREAM`, where the user must set the stream callback functions themselves.

The three callback functions are invoked to open, close, and read from the input source. The functions should have been declared using the function prototype macros `XML_OPENF`, `XML_CLOSEF` and `XML_READF`.

`XML_OPENF` is the open function, called once to open the input source. It should set its persistent handle in the `xmlihdl` union, which has two choices, a generic pointer `(void *)`, and an integer (as unix file or socket handle). This function must return `XMLERR_OK` on success.

Parameter	IN / OUT	Description
ctx	(IN)	XML context.
ih	(OUT)	The opened handle is placed here.
length	(OUT)	Total length of input data in bytes, if known (0 if not known).
parts	(IN)	URL broken down into components; opaque pointer.
path	(IN)	Full URL to be opened.

`XML_CLOSEF` is the close function; it closes an open source and frees resources.

Parameter	IN / OUT	Description
ctx	(IN)	XML context.
ih	(IN)	The opened handle.

`XML_READF` is the reader function; it reads data from an open source into a buffer, and returns the number of bytes read:

- If <= 0, an EOI condition is indicated.
- If > 0, then the EOI flag determines if this is the final data.

On EOI, the matching close function will be called automatically.

Parameter	IN / OUT	Description
ctx	(IN)	XML context.
dest	(OUT)	Destination buffer to read data into.
destsize	(IN)	Size of dest buffer.
eoi	(OUT)	Hit End of Information?
ih	(IN)	Input handle union from which to read.
nraw	(OUT)	Number of bytes read.
path	(IN)	URL of source provided for use in error messages.

xmlclean()

Description

Recycles memory within the XML parser, but does not free it to the system; only `xmlterm()` finally releases all memory back to the system. If `xmlclean()` is not called between parses, then the data used by the previous documents remains allocated, and pointers to it are valid. Thus, the data for multiple documents can be accessible simultaneously, although only the current document can be manipulated with DOM.

If only access to only one document's data at a time within a single context is desired, than `clear()` should be called before each new parse.

Syntax

```
void xmlclean( xmlctx *ctx);
```

Parameter	IN / OUT	Description
ctx	(IN)	The XML parser context

xmlinit()

Description

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, nonzero 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.
- 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 data encoding is UTF-8. If documents are singlebyte, encoding should be set accordingly for better performance.
- Messages are printed to `stderr` unless `msghdlr` is given.

- By default, a DOM tree is built. For SAX mode, the `saxcb` callbacks should be set. Note that any of the SAX callback functions may be set to `NULL` if not needed.
- The memory callback `memcb` may be used for user-defined memory allocation. Both `alloc()` and `free()` functions must be specified.
- The parameters `msgctx`, `saxcbctx`, and `memcbctx` can be used to define and pass information to user-defined callback routines for the message handler, SAX functions, or memory functions, respectively. They should be set to `NULL` if user-defined callback functions do not need any additional information passed in to them.
- The `lang` parameter determines the language of error messages; the default is “American.”
- Returns the XML context on success; sets the user’s `err` argument on error. A zero error code means success, nonzero indicates a problem.

Syntax

```
xmlctx *xmlinit( uword *err,
                 const oratext *incoding,
                 XML_MSGHDLRF((*msghdlr)),
                 void *msgctx,
                 const xmlsaxcb *saxcb,
                 void *saxcbctx,
                 const xmlmemcb *memcb,
                 void *memcbctx,
                 const oratext *lang);
```

Parameter	IN / OUT	Description
err	(OUT)	Numeric error code, on failure.
incoding	(IN)	Default input encoding.
msghdlr	(IN)	Error message handler function.
msgctx	(IN)	User’s context for the error message handler.
saxcb	(IN)	SAX callback structure, filled with function pointers.
saxcbctx	(IN)	User’s context for SAX callbacks.
memcb	(IN)	Memory function callback structure.
memcbctx	(IN)	User’s context for the memory function callbacks.
lang	(IN)	Language for error messages.

Error Code	Description
XMLERR_BAD_ENCODING	An encoding was not known. Use IANA / Mine names for encodings, and make sure Globalization Support data is present.
XMLERR_INVALID_LANG	The language specified for error messages was not known.
XMLERR_INVALID_MEMCB	A memory callback structure (memcb) was specified, but it did not have alloc and free function pointers.
XMLERR_LEH_INIT	The LEH (catch / throw) package could not be initialized. An internal error, contact support.

Error Code	Description
XMLERR_NLS_INIT	The National Language Service package could not be initialized. Perhaps an installation or configuration problem.

xmlinitenc()

Description

Initializes the XML parser, specifying DOM data encoding. It must be called before any parsing can take place. Same as SAX `xmlinit()`, but allows data encoding to be specified.

Syntax

```
xmlctx *xmlinitenc( uword *err,
                    const oratext *incoding,
                    const oratext *outcoding,
                    XML_MSGHDLRF((*msghdlr)),
                    void *msgctx,
                    const xmlsaxcb *saxcb,
                    void *saxcbctx,
                    const xmlmemcb *memcb,
                    void *memcbctx,
                    const oratext *lang);
```

Parameter	IN / OUT	Description
err	(OUT)	Numeric error code, on failure.
incoding	(IN)	Default input encoding.
outcoding	(IN)	Output (DIM/SAX data) character set encoding.
msghdlr	(IN)	Error message handler function.
msgctx	(IN)	User’s context for the error message handler.
saxcb	(IN)	SAX callback structure, filled with function pointers.
saxcbctx	(IN)	User’s context for SAX callbacks.
memcb	(IN)	Memory function callback structure.
memcbctx	(IN)	User’s context for the memory function callbacks.
lang	(IN)	Language for error messages.

Error Code	Description
XMLERR_BAD_ENCODING	An encoding was not known. Use IANA / Mine names for encodings, and make sure Globalization Support data is present.
XMLERR_INVALID_LANG	The language specified for error messages was not known.
XMLERR_INVALID_MEMCB	A memory callback structure (memcb) was specified, but it did not have alloc and free function pointers.
XMLERR_LEH_INIT	The LEH (catch / throw) package could not be initialized. An internal error, contact support.
XMLERR_NLS_INIT	The National Language Service package could not be initialized. Perhaps an installation or configuration problem.

Parameter	IN / OUT	Description

xmlLocation()

Description

Returns current source location while parsing. This function may be called at any time. However, a `0` will be returned for both `path` and `line` if the call is not made during parsing.

Syntax

```
uword xmlLocation( xmlctx *ctx,
                   ub4 *line,
                   oratext **path);
```

Parameter	IN / OUT	Description
ctx	(IN)	The XML parser context
line	(OUT)	Current line number
path	(OUT)	Current source path / URL

xmlparse()

Description

Invokes the XML parser on an input document that is specified by a URI. The parser must have been initialized successfully with a call to `xmlinit()` or `xmlinitenc()` first. Parser options are specified as flag bits OR'd together into the `flags` mask.

The default input encoding may be specified as `incoding`, which overrides the `incoding` given to `xmlinit()`. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be `incoding`. IANA / Mime encoding names should be used, “`UTF-8`”, “`ASCII`”, others.

Syntax

```
uword xmlparse( xmlctx *ctx,
                const oratext *uri,
                const oratext *incoding,
                ub4 flags);
```

Parameter	IN / OUT	Description
ctx	(IN/OUT)	The XML parser context
uri	(IN)	URI of XML document
incoding	(IN)	Default input encoding
flags	(IN)	Mask of parser option flag bits

Parser Flag Option	Description
XML_DTD_ONLY	Parses an external subset. This is the same as calling `xmlparsedtd()`; it parses an external subset (DTD) instead of a complete XML document. Used primarily by the Class Generator so that it may generate classes from a DTD without need of a complete document.

Parser Flag Option	Description
XML_FLAG_DISCARD_WHITESPACE	Discards extraneous whitespace (end-of-line, and so on); default behavior is to report whitespace and indicate which whitespace can be ignored. This option will discard all whitespace between an end-element tag and the following start-element tag.
XML_FLAG_STOP_ON_WARNING	Stops validation on warnings. Validation problems are considered warnings (non-fatal) unless this flag is set. If set, validation will stop after the first warning.
XML_FLAG_VALIDATE	Turns validation on; default behavior is to only check for well-formedness.
XML_FORCE_INCODING	Forces input documents to be interpreted in the encoding "incoding". The default input encoding may be specified as incoding, which overrides the incoding given to xmlinit. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be incoding. IANA / Mime encoding names should be used, "UTF-8", "ASCII", and so on. If `XML_FLAG_FORCE_INCODING` is set, the document will be interpreted as "incoding" regardless.
XML_WARN_DUPLICATE_ENTITY	Causes a warning to be emitted if a duplicate entity declaration is found. A duplicate entity declaration is usually silently ignored. When set, this flag causes a warning to be emitted instead.

xmlparsebuf()

Description

Invokes the XML parser on a buffer. The parser must have been initialized successfully with a call to `xmlinit()` or `xmlinitenc()` first. Parser options are specified as flag bits OR'd together into the `flags` mask.

The default input encoding may be specified as `incoding`, which overrides the `incoding` given to `xmlinit()`. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be `incoding`. IANA / Mime encoding names should be used, "`UTF-8`", "`ASCII`", others.

Syntax

```
uword xmlparsebuf( xmlctx *ctx,
                   const oratext *buffer,
                   size_t len,
                   const oratext *incoding,
                   ub4 flags);
```

Parameter	IN / OUT	Description
ctx	(IN/OUT)	The XML parser context
buffer	(IN)	Input buffer name
len	(IN)	Length of the buffer
incoding	(IN)	Default input encoding
flags	(IN)	Mask of parser option flag bits

Parser Flag Option	Description
XML_DTD_ONLY	Parses an external subset. This is the same as calling `xmlparsedtd()`; it parses an external subset (DTD) instead of a complete XML document. Used primarily by the Class Generator so that it may generate classes from a DTD without need of a complete document.
XML_FLAG_DISCARD_WHITESPACE	Discards extraneous whitespace (end-of-line, and so on); default behavior is to report whitespace and indicate which whitespace can be ignored. This option will discard all whitespace between an end-element tag and the following start-element tag.
XML_FLAG_STOP_ON_WARNING	Stops validation on warnings. Validation problems are considered warnings (non-fatal) unless this flag is set. If set, validation will stop after the first warning.
XML_FLAG_VALIDATE	Turns validation on; default behavior is to only check for well-formedness.

Parser Flag Option	Description
XML_FORCE_INCODING	Forces input documents to be interpreted in the encoding “incoding”. The default input encoding may be specified as incoding, which overrides the incoding given to xmlinit. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be incoding. IANA / Mime encoding names should be used, “UTF-8”, “ASCII”, and so on. If `XML_FLAG_FORCE_INCODING` is set, the document will be interpreted as “incoding” regardless.
XML_WARN_DUPLICATE_ENTITY	Causes a warning to be emitted if a duplicate entity declaration is found. A duplicate entity declaration is usually silently ignored. When set, this flag causes a warning to be emitted instead.

xmlparsedtd()

Description

Invokes the XML parser on an external DTD file, not a complete document. It is used mainly by the Class Generator to create classes from a DTD without requiring a complete document. The parser must have been initialized successfully with a call to `xmlinit()` or `xmlinitenc()` first. Parser options are specified as flag bits OR'd together into the `flags` mask.

The default input encoding may be specified as `incoding`, which overrides the `incoding` given to `xmlinit()`. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be `incoding`. IANA / Mime encoding names should be used, “`UTF-8`”, “`ASCII`”, others.

Syntax

```
uword xmlparsedtd( xmlctx *ctx,
                   const oratext *filename,
                   oratext *name,
                   const oratext *incoding,
                   ub4 flags);
```

Parameter	IN / OUT	Description
ctx	(IN/OUT)	The XML parser context
filename	(IN)	File name of the external subset
name	(IN)	Name of the DTD
incoding	(IN)	Default input encoding
flags	(IN)	Mask of parser option flag bits

Parser Flag Option	Description
XML_DTD_ONLY	Parses an external subset. This is the same as calling xmlparsedtd; it parses an external subset (DTD) instead of a complete XML document. Used primarily by the Class Generator so that it may generate classes from a DTD without need of a complete document.
XML_FLAG_DISCARD_WHITESPACE	Discards extraneous whitespace (end-of-line, and so on); default behavior is to report whitespace and indicate which whitespace can be ignored. This option will discard all whitespace between an end-element tag and the following start-element tag.
XML_FLAG_STOP_ON_WARNING	Stops validation on warnings. Validation problems are considered warnings (non-fatal) unless this flag is set. If set, validation will stop after the first warning.
XML_FLAG_VALIDATE	Turns validation on; default behavior is to only check for well-formedness.
XML_FORCE_INCODING	Forces input documents to be interpreted in the encoding “incoding”. The default input encoding may be specified as incoding, which overrides the incoding given to xmlinit. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl,and others, then it is assumed to be incoding. IANA / Mime encoding names should be used, “UTF-8”, “ASCII”, and so on. If `XML_FLAG_FORCE_INCODING` is set, the document will be interpreted as “incoding” regardless.
XML_WARN_DUPLICATE_ENTITY	Causes a warning to be emitted if a duplicate entity declaration is found. A duplicate entity declaration is usually silently ignored. When set, this flag causes a warning to be emitted instead.

xmlparsefile()

Description

Invokes the XML parser on a document in the file system. The parser must have been initialized successfully with a call to `xmlinit()` or `xmlinitenc()` first. Parser options are specified as flag bits OR'd together into the `flags` mask.

The default input encoding may be specified as `incoding`, which overrides the `incoding` given to `xmlinit()`. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be `incoding`. IANA / Mime encoding names should be used, “`UTF-8`”, “`ASCII`”, others.

Syntax

```
uword xmlparsefile( xmlctx *ctx,
                    const oratext *path,
                    const oratext *incoding,
                    ub4 flags);
```

Parameter	IN / OUT	Description
ctx	(IN/OUT)	The XML parser context
path	(IN)	Filesystem path of the document
incoding	(IN)	Default input encoding
flags	(IN)	Mask of parser option flag bits

Parser Flag Option	Description
XML_DTD_ONLY	Parses an external subset. This is the same as calling xmlparsedtd; it parses an external subset (DTD) instead of a complete XML document. Used primarily by the Class Generator so that it may generate classes from a DTD without need of a complete document.
XML_FLAG_DISCARD_WHITESPACE	Discards extraneous whitespace (end-of-line, and so on); default behavior is to report whitespace and indicate which whitespace can be ignored. This option will discard all whitespace between an end-element tag and the following start-element tag.

Parser Flag Option	Description
XML_FLAG_STOP_ON_WARNING	Stops validation on warnings. Validation problems are considered warnings (non-fatal) unless this flag is set. If set, validation will stop after the first warning.
XML_FLAG_VALIDATE	Turns validation on; default behavior is to only check for well-formedness.
XML_FORCE_INCODING	Forces input documents to be interpreted in the encoding “incoding”. The default input encoding may be specified as incoding, which overrides the incoding given to xmlinit. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl,and others, then it is assumed to be incoding. IANA / Mime encoding names should be used, “UTF-8”, “ASCII”, and so on. If `XML_FLAG_FORCE_INCODING` is set, the document will be interpreted as “incoding” regardless.
XML_WARN_DUPLICATE_ENTITY	Causes a warning to be emitted if a duplicate entity declaration is found. A duplicate entity declaration is usually silently ignored. When set, this flag causes a warning to be emitted instead.

xmlparsestream()

Description

Invokes the XML parser on a user-defined stream. The parser must have been initialized successfully with a call to `xmlinit()` or `xmlinitenc()` first. Parser options are specified as flag bits OR'd together into the `flags` mask to override the default behavior of the parser.

The default input encoding may be specified as `incoding`, which overrides the `incoding` given to `xmlinit()`. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be `incoding`. IANA / Mime encoding names should be used, “`UTF-8`”, “`ASCII`”, others.

In the context of this function, a stream is a user defined entity; `stream` is a stream / context pointer, which is in turn passed to the I / O callback functions. The parser does not reference the stream directly. 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 oratext *stream,
                      const oratext *incoding,
                      ub4 flags);
```

Parameter	IN / OUT	Description
ctx	(IN/OUT)	The XML parser context
stream	(IN)	Pointer to a user-defined stream object
incoding	(IN)	Default input encoding
flags	(IN)	Mask of parser option flag bits

Parser Flag Option	Description
XML_DTD_ONLY	Parses an external subset. This is the same as calling xmlparsedtd; it parses an external subset (DTD) instead of a complete XML document. Used primarily by the Class Generator so that it may generate classes from a DTD without need of a complete document.
XML_FLAG_DISCARD_WHITESPACE	Discards extraneous whitespace (end-of-line, and so on); default behavior is to report whitespace and indicate which whitespace can be ignored. This option will discard all whitespace between an end-element tag and the following start-element tag.
XML_FLAG_STOP_ON_WARNING	Stops validation on warnings. Validation problems are considered warnings (non-fatal) unless this flag is set. If set, validation will stop after the first warning.
XML_FLAG_VALIDATE	Turns validation on; default behavior is to only check for well-formedness.

Parser Flag Option	Description
XML_FORCE_INCODING	Forces input documents to be interpreted in the encoding "incoding". The default input encoding may be specified as incoding, which overrides the incoding given to xmlinit. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl,and others, then it is assumed to be incoding. IANA / Mime encoding names should be used, "UTF-8", "ASCII", and so on. If `XML_FLAG_FORCE_INCODING` is set, the document will be interpreted as "incoding" regardless.
XML_WARN_DUPLICATE_ENTITY	Causes a warning to be emitted if a duplicate entity declaration is found. A duplicate entity declaration is usually silently ignored. When set, this flag causes a warning to be emitted instead.

xmlterm()

Description

Terminates the XML parser. It should be called after `xmlinit`, and before exiting the main program.

Syntax

```
uword xmlterm(xmlctx *ctx);
```

Parameter	IN / OUT	Description
ctx	(IN/OUT)	The XML parser context

Comments

This function terminates an XML context. All memory used by the parser is returned to the system. The context may not be reused; instead, a new context must be created if additional parsing is to be done. No additional XML parser calls can be made until `xmlinit` is called again to get a new context. Compare to `xmlclean()`, which recycles memory internally without giving it back to the system, and allows the context to continue to be used.

xmlwhere()

Description

Returns error location information for the last, or current, error. Returns source location where the current error occurred. Should be called from within an error handler. The source location is a stack. Index `0` is the lowest level where the problem actually occurred, index `1` is the enclosing entity, and so on. To show the whole stack, the index should be looped form `0` to `N`, stopping when `FALSE` is returned.

Syntax

```
boolean xmlwhere( xmlctx *ctx,
                  ub4 *line,
                  oratext **path,
                  unword idx);
```

Parameter	IN / OUT	Description
ctx	(IN)	The XML parser context
line	(OUT)	Line number where the error occurred
path	(OUT)	Source path / URL where the error occurred
idx	(IN)	Error number in stack, starting at `0`

W3C SAX APIs

Description of W3C SAX APIs

SAX is a standard interface for event-based XML parsing, developed cooperatively by the members of the XML-DEV mailing list.

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.

Callback Structure of W3C SAX APIs

```
typedef struct
{
   sword (*startDocument)(void *ctx);
   sword (*endDocument)(void *ctx);
   sword (*startElement)(void *ctx, const oratext *name,
                         const struct xmlnodes *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);
   sword (*comment)(void *ctx, const oratext *data);
   sword (*elementDecl)(void *ctx, const oratext *name,
                         const oratext *content);
   sword (*attributeDecl)(void *ctx, const oratext *elem,
                         const oratext *attr, const oratext *body);
   sword (*xmlDecl)(void *ctx, const oratext *version, boolean encoding);
} xmlsaxcb;
```

Data Structures of W3C SAX APIs

Table 13–3 Data Structures Namespace APIs

Data Stricture	Declaration	Description
oratext*	typedef unsigned char oratext;	Pointer to data; any encoding, including Unicode; cast as needed
sword	typedef signed int sword;	Return value for user SAX callback functions; `0` means success.
xmlnodes*	typedef struct xmlnodes xmlnodes;	Pointer to list of nodes (attributes of an element). Note: `xmlnodes*` is an opaque type and cannot be referenced directly. Instead, use DOM function getAttributeIndex()

Callback Functions

Table 13–4 Summary of SAX Callback Functions

Function	Description
characters() on page 13-31	Receives notification of character data inside an element.
endDocument() on page 13-31	Receives notification of the end of the document.
endElement() on page 13-32	Receives notification of the end of an element.
ignorableWhitespace() on page 13-32	Receives notification of ignorable whitespace in content.
notationDecl() on page 13-33	Receives notification of a notation declaration.
processingInstruction() on page 13-33	Receives notification of a processing instruction.
startDocument() on page 13-34	Receives notification of the beginning of the document.
startElement() on page 13-34	Receives notification of the start of an element.
unparsedEntityDecl() on page 13-34	Receives notification of an unparsed entity declaration.

Table 13–5 Summary of Oracle SAX Extension Callback Functions

Function	Description
attributeDecl() on page 13-35	Receive notification about an element attribute declaration in the DTD

Table 13–5 Summary of Oracle SAX Extension Callback Functions

Function	Description
comment() on page 36	Receive notification about a comment in the XML document
elementDecl() on page 13-36	Receive notification about an element declaration in the DTD
ignorableWhitespace() on page 13-32	Receive notification of the start of a namespace for an element.

characters()

Description

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)	User's context
ch	(IN)	Pointer to character data
len	(IN)	Length of data in characters

endDocument()

Description

Receives notification of the end of the document.

Syntax

```
sword (*endDocument) void *ctx);
```

Parameter	IN / OUT	Description
ctx	(IN)	client context

endElement()

Description

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()

Description

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)	User's context
ch	(IN)	Pointer to character data
len	(IN)	Length of data in characters

notationDecl()

Description

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)	User's context
name	(IN)	Notation name
publicID	(IN)	Notation public identifier, `NULL` if not available
systemId	(IN)	Notation system identifier

processingInstruction()

Description

Receives notification of a processing instruction.

Syntax

```
sword (*processingInstruction)( void *ctx,
                                const oratext *target,
                                const oratext *data);
```

Parameter	IN / OUT	Description
ctx	(IN)	User's context
target	(IN)	Processing instruction target
data	(IN)	Processing instruction data; NULL if no data is supplied

startDocument()

Description

Receives notification of the beginning of the document.

Syntax

```
sword (*startDocument)( void *ctx);
```

Parameter	IN / OUT	Description
ctx	(IN)	User’s context

startElement()

Description

Receives non-namespace-aware notification of the beginning of an element.

Syntax

```
sword (*startElement)( void *ctx,
                       const oratext *name,
                       const struct xmlnodes *attrs);
```

Parameter	IN / OUT	Description
ctx	(IN)	User’s context
name	(IN)	Element name
attrs	(IN)	Specified or defaulted attributes

unparsedEntityDecl()

Description

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)	User’s context
name	(IN)	Entity name
publicId	(IN)	Entity public identifier, `NULL` if not available
systemId	(IN)	Entity system identifier
notationName	(IN)	Name of the associated notation

attributeDecl()

Description

Receives notification about an element's attribute declaration.

Syntax

```
sword (*attributeDecl)( void *ctx,
                        const oratext *elem,
                        const oratext *attr,
                        const oratext *body);
```

Parameter	IN / OUT	Description
ctx	(IN)	User’s context
elem	(N)	Element name
attr	(IN)	Attribute name
body	(IN)	Body of attribute declaration

comment()

Description

Receives notification about a comment.

Syntax

```
sword (*comment)( void *ctx,
                  const oratext *data);
```

Parameter	IN / OUT	Description
ctx	(IN)	client context
data	(IN)	body of comment

elementDecl()

Description

Receives notification about an element declaration.

Syntax

```
sword (*elementDecl)( void *ctx,
                      const oratext *name,
                      const oratext *content);
```

Parameter	IN / OUT	Description
ctx	(IN)	User’s context
name	(IN)	Element name
content	(IN)	Element content model

nsStartElement()

Description

Receives namespace-aware notification of the start of an element.

Syntax

```
sword (*nsStartElement)( void *ctx,
                         const oratext *qname,
                         const oratext *local,
                         const oratext *namespace,
                         const struct xmlnodes *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

Description of W3C DOM APIs

Since the DOM standard is object-oriented, the following changes were made for the C adaptation:

- Reused function names were expanded; for example, `getValue()` in the attribute class was given the unique name `getAttrValue()`, matching the pattern established by `getNodeValue()`.
- Some functions were added to extend the DOM; for example, there is no function defined that returns the number of children of a node, so `numChildNodes()` was added.

The implementation of this C DOM interface follows REC-DOM-Level-1-19981001.

Data Structures of W3C DOM APIs

Table 13–6 Data Structures of W3C DOM APIs

Data Structure	Declaration	Description
boolean	typedef int boolean;	Boolean value, TRUE or FALSE
oratext	typedef unsigned char oratext;	String pointer
xmlctx	typedef struct xmlctx xmlctx;	Master XML parser context. Note: The contents of xmlctx are private and must not be accessed by users.
xmlnode	typedef struct xmlnode xmlnode;	Document node. Note: The contents of xmlnode are private and must not be accessed by users.
xmlnodes	typedef struct xmlnodes xmlnodes;	Array of nodes. Note: The contents of xmlnodes are private and must not be accessed by users.

Table 13–6 Data Structures of W3C DOM APIs (Cont.)

Data Structure	Declaration	Description
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 = /* document */ DOCUMENT_TYPE_NODE = 10 /* DTD */ DOCUMENT_FRAGMENT_NODE = 11 /* document fragment */ NOTATION_NODE = 12 /* notation */	Node type enumeration Note: Names and values match DOM specification. Parse tree node types; see `getNodeType()`.

Functions of W3C DOM APIs

Table 13–7 Summary of Functions of W3C DOM APIs

Function	Description
appendChild() on page 13-43	Appends child node to the parent's list of children.
appendData() on page 13-44	Appends character data to the node's current data.
cloneNode() on page 13-44	Creates a new node identical to the given node.
createAttribute() on page 13-45	Creates a non-namespace aware attribute for an element node.
createAttributeNS() on page 13-45	Creates a namespace-aware attribute for an element node.
createCDATASection() on page 13-46	Creates a `CDATA_SECTION` node.
createComment() on page 13-46	Creates a `COMMENT` node.

Table 13–7 Summary of Functions of W3C DOM APIs (Cont.)

Function	Description
createDocument() on page 13-47	Creates a non-namespace-aware DOCUMENT node
createDocumentFragment() on page 13-47	Creates a DOCUMENT_FRAGMENT node.
createDocumentNS() on page 13-48	Creates a namespace-aware DOCUMENT node.
createDocumentType() on page 13-48	Creates a new DTD node.
createElement() on page 13-49	Creates a non-namespace-aware ELEMENT node.
createElementNS() on page 13-49	Creates a namespace-aware ELEMENT node.
createEntityReference() on page 13-50	Creates an ENTITY_REFERENCE node.
createProcessingInstruction() on page 13-50	Creates a PROCESSING_INSTRUCTION (PI) node.
createTextNode() on page 13-51	Creates a TEXT node.
deleteData() on page 13-51	Removes substring from a node's character data.
getAttribute() on page 13-52	Returns one attribute from an array, given its index.
getAttributeIndex() on page 13-52	Returns an element's attribute given its index.
getAttrName() on page 13-53	Returns an attribute's name.
getAttributeNode() on page 13-53	Returns an element's attribute node given its name.
getAttributes() on page 13-54	Returns an array of element's attributes.
getAttrSpecified() on page 13-54	Returns value of attribute's specified flag.
getAttrValue() on page 13-54	Returns the value of an attribute
getCharData() on page 13-55	Returns character data for a TEXT node.
getCharLength() on page 13-55	Returns length of TEXT node's character data.
getChildNode() on page 13-56	Returns node from array of nodes, given the child’s index.

Table 13–7 Summary of Functions of W3C DOM APIs (Cont.)

Function	Description
getChildNodes() on page 13-56	Returns array of node's children.
getContentModel() on page 13-56	Returns the content model for an element from the DTD.
getDocOrder() on page 13-57	Returns a document order cardinal for a node.
getDocType() on page 13-57	Returns current DTD.
getDocTypeEntities() on page 13-58	Returns a list of a DTD's general entities.
getDocTypeName() on page 13-58	Returns name of DTD.
getDocTypeNotations() on page 13-58	Returns a list of DTD's notations.
getDocument() on page 13-59	Returns top-level `DOCUMENT` node.
getDocumentElement() on page 13-59	Returns highest-level, or root, `ELEMENT` node.
getElementByID() on page 13-59	Returns the element node which has the given ID.
getElementsByTagName() on page 13-60	Returns list of elements with matching name.
getElementsByTagNameNS() on page 13-60	Returns list of elements with matching name with namespace information.
getEntityNotation() on page 13-61	Returns an entity's NDATA.
getEntityPubID() on page 13-61	Returns an entity's public ID.
getEntitySysID() on page 13-62	Returns an entity's system ID.
getFirstChild() on page 13-62	Returns the first child of a node
getImplementation() on page 13-62	Returns DOM-implementation structure.
getLastChild() on page 13-63	Returns the last child of a node.
getNamedItem() on page 13-63	Returns the named node from a list of nodes.
getNextSibling() on page 13-64	Returns a node's next sibling.
getNodeMapLength() on page 13-64	Returns number of entries in a NodeMap [DOM getLength]

Table 13–7 Summary of Functions of W3C DOM APIs (Cont.)

Function	Description
getNodeName() on page 13-64	Returns a node's name
getNodeType() on page 13-65	Returns a node's type code (enumeration)
getNodeValue() on page 13-65	Returns a node's "value", its character data
getNotationPubID() on page 13-66	Returns a notation's public ID [DOM getPublicId]
getNotationSysID() on page 13-66	Returns a notation's system ID [DOM getSystemId]
getOwnerDocument() on page 13-66	Returns the DOCUMENT node containing the given node
getParentNode() on page 13-67	Returns a node's parent node
getPIData() on page 13-67	Returns a processing instruction's data [DOM getData]
getPITarget() on page 13-67	Returns a processing instruction's target [DOM getTarget]
getPreviousSibling() on page 13-68	Returns a node's "previous" sibling
getTagName() on page 13-68	Returns a node's "tagname", same as name for now
hasAttributes() on page 13-69	Determine if element node has attributes [DOM extension]
hasChildNodes() on page 13-69	Determine if node has children
hasFeature() on page 13-69	Determine if DOM implementation supports a specific feature
importNode() on page 13-70	Copy a node from a different document into this document
insertBefore() on page 13-71	Inserts a new child node before the given reference node
insertData() on page 13-71	Inserts new character data into a node's existing data
isStandalone() on page 13-72	Determine if document is standalone [DOM extension]
nodeValid() on page 13-72	Validate a node against the current DTD [DOM extension]
normalize() on page 13-73	Normalize a node by merging adjacent TEXT nodes
numAttributes() on page 13-73	Returns number of element node's attributes [DOM extension]
numChildNodes() on page 13-74	Returns number of node's children [DOM extension]
prefixToURI() on page 13-74	Returns a matching URI, given a namespace prefix and a node.
removeAttribute() on page 13-74	Removes an element's attribute given its names

Table 13–7 Summary of Functions of W3C DOM APIs (Cont.)

Function	Description
removeAttributeNode() on page 13-75	Removes an element's attribute given its pointer
removeChild() on page 13-75	Removes a node from its parents list of children
removeNamedItem() on page 13-76	Removes a node from a list of nodes given its name
replaceChild() on page 13-76	Replace one node with another
replaceData() on page 13-76	Replace a substring of a node's character data with another string
saveString() on page 13-77	Saves a character string in the XML memory pool
saveString2() on page 13-77	
setAttribute() on page 13-78	Sets (adds or replaces) a new attribute for an element node given the attribute's name and value
setAttributeNode() on page 13-79	Sets (adds or replaces) a new attribute for an element node given a pointer to the new attribute
setAttrValue() on page 13-79	Sets an attribute's value
setCharData() on page 13-80	Sets a TEXT or CDATA node's value
setNamedItem() on page 13-80	Sets (adds or replaces) a new node in a parent's list of children
setNodeValue() on page 13-81	Sets a node's "value" (character data)
setPIData() on page 13-81	Sets a processing instruction's data [DOM setData]
splitText() on page 13-81	Split a node's character data into two parts
substringData() on page 13-82	Return a substring of a node's character data

appendChild()

Description

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

appendData()

Description

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()

Description

Returns a duplicate of this node; serves as a generic copy constructor for nodes. The duplicate node has no parent, therefore `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 is different in that the node's children are also recursively cloned, instead of just being 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()

Description

Creates 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

createAttributeNS()

Description

Creates a new `ATTRIBUTE` node with the given name and value with namespace information. The new node is unattached and must be added to an element node with `setAttributeNode()`.

Syntax

```
xmlnode *createAttributeNS( xmlctx *ctx,
                            const oratext *uri,
                            const oratext *qname,
                            const oratext *value);
```

Parameter	IN / OUT	Description
ctx	(IN)	xml context
uri	(IN)	namespace URI of new attribute
qname	(IN)	qualified name of new attribute
value	(IN)	value of new attribute

createCDATASection()

Description

Creates a new `CDATA` node.

Syntax

```
xmlnode *createCDATASection( xmlctx *ctx,
                             const oratext *data);
```

Parameter	IN / OUT	Description
ctx	(IN)	xml context
data	(IN)	CDATA body

createComment()

Description

Creates 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

createDocument()

Description

Creates a new document in memory. The original function `createDocument()` has now been standardized in DOM 2.0 CORE. For compatibility, the old function remains with its original usage, and the new CORE function is called `createDocumentNS()`. 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

createDocumentFragment()

Description

Creates 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

createDocumentNS()

Description

Creates a new document in memory. The original function `createDocument()` has now been standardized in DOM 2.0 CORE. For compatibility, the old function remains with its original usage, and the new CORE function is called createDocumentNS. Creates 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`. If a DTD is specified, its `ownerDocument()` attribute will be set to the document being created.

Syntax

```
xmlnode* createDocumentNS( xmldomimp *imp,
                           oratext *uri,
                           oratext *qname,
                           xmlnode *dtd);
```

Parameter	IN / OUT	Description
imp	(IN)	XML DOMImplementation, see `getImplementation()`
uri	(IN)	new document's namespace URI
qname	(IN)	namespace qualified name of new document; `DOCUMENT_NODE`'s name
dtd	(IN)	DTD with which this document is associated

createDocumentType()

Description

Creates a new document type (DTD) node.

Syntax

```
xmlnode* createDocumentType( xmldomimp *imp,
                             oratext *qname,
                             oratext *pubid,
                             oratext *sysid);
```

Parameter	IN / OUT	Description
imp	(IN)	The XML DOM implementation; see `getImplementation()`.
pubid	(IN)	External subset public identifier
qname	(IN)	The namespace qualified name of a new document type; `DOCUMENT_NODE`'s name
sysid	(IN)	External subset system identifier

createElement()

Description

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

createElementNS()

Description

Creates a new `ELEMENT` node with namespace information.

Syntax

```
xmlnode *createElementNS( xmlctx *ctx,
                            const oratext *uri,
                            const oratext *qname);
```

Parameter	IN / OUT	Description
ctx	(IN)	xml context
uri	(IN)	namespace URI of new element
qname	(IN)	qualified name of new element

createEntityReference()

Description

Creates 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 to reference

createProcessingInstruction()

Description

Creates 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()

Description

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()

Description

Deletes 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

Parameter	IN / OUT	Description
offset	(IN)	offset of start of substring; `0` is first char
count	(IN)	length of substring

getAttribute()

Description

Returns one attribute from an array of attributes, given an index (starting at `0`). Fetch the attribute name or value, or both, with `getAttrName()` and `getAttrValue()`. On error, returns `NULL`.

Syntax

```
const oratext *getAttribute( const xmlnode *node,
                             const oratext *name);
```

Parameter	IN / OUT	Description
node	(IN)	pointer to node
name	(IN)	name of attribute

getAttributeIndex()

Description

Returns one attribute from an array of attributes, given an index (starting at 0). Fetches the attribute name or value, or both, 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 nodes structure, as returned by `getAttribute()`

Parameter	IN / OUT	Description
index	(IN)	zero-based attribute number returned

getAttrName()

Description

Returns the name of the attribute given a pointer to that 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; see `getAttribute()`

getAttributeNode()

Description

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()

Description

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)	node whose attributes to return

getAttrSpecified()

Description

Returns 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; see `getAttribute()`

getAttrValue()

Description

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; see `getAttribute()`

getCharData()

Description

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 text node

getCharLength()

Description

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 text node

getChildNode()

Description

Returns the nth 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; see `getChildNodes()`
index	(IN)	zero-based child number

getChildNodes()

Description

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 to return

getContentModel()

Description

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.

Syntax

```
xmlnode *getContentModel( xmldtd *dtd,
                          oratext *name);
```

Parameter	IN / OUT	Description
dtd	(IN)	pointer to the DTD
name	(IN)	name of element

getDocOrder()

Description

Returns the document order cardinal for a node. setDocOrder() must have been called first or all nodes will have a 0 order. This function is used primarily by the XSLT processor.

Syntax

```
ub4 getDocOrder( xmlnode *node);
```

Parameter	IN / OUT	Description
node	(IN)	Node whose doc order to return.

getDocType()

Description

Returns a pointer to the (opaque) DTD for the current document.

Syntax

```
xmldtd* getDocType( xmlctx *ctx);
```

Parameter	IN / OUT	Description
ctx	(IN)	XML parser content

getDocTypeEntities()

Description

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()

Description

Returns the given DTD's name.

Syntax

```
oratext *getDocTypeName( xmldtd* dtd);
```

Parameter	IN / OUT	Description
dtd	(IN)	pointer to the DTD

getDocTypeNotations()

Description

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

getDocument()

Description

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()

Description

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

getElementByID()

Description

Returns the element node which has the given ID. If no such ID is defined (or other problems), returns `NULL.`

Syntax

```
xmlnode *getElementByID( xmlctx *ctx,
                                oratext *id);
```

Parameter	IN / OUT	Description
ctx	(IN)	XML parser context
id	(IN)	element id

getElementsByTagName()

Description

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 of tree
name	(IN)	element tag name

getElementsByTagNameNS()

Description

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 *getElementsByTagNameNS( xmlctx *ctx,
                                   xmlnode *root,
                                   const oratext *uri,
                                   const oratext *local);
```

Parameter	IN / OUT	Description
ctx	(IN)	XML parser context
root	(IN)	root node of tree
uri	(IN)	element namespace uri
local	(IN)	element local name

getEntityNotation()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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 node to fetch
index	(OUT)	index of found node

Parameter	IN / OUT	Description
nodes	(IN)	array of nodes

getNextSibling()

Description

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()

Description

Returns the number of nodes in the map, given an array of nodes returned by `getChildNodes()`. 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()

Description

Returns the name of the given node, or NULL if the node has no name. Note that “tagname” and “name” are currently synonymous.

Syntax

```
const oratext* getNodeName(const xmlnode *node);
```

Parameter	IN / OUT	Description
node	(IN)	pointer to node

getNodeType()

Description

Returns the type code for a node.

Syntax

```
xmlntype getNodeType( const xmlnode *node);
```

Parameter	IN / OUT	Description
node	(IN)	pointer to node

getNodeValue()

Description

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()

Description

Return a notation node's public ID. Under the DOM spec, this is a method named `getPublicId()`.

Syntax

```
const oratext *getNotationPubID( const xmlnode *node);
```

Parameter	IN / OUT	Description
node	(IN)	pointer to node

getNotationSysID()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

Returns the previous sibling of the given node; 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()

Description

Returns the “tagname” of a node, which is the same as its name for now, see getNodeName. The DOM specification states that “...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()

Description

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()

Description

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()

Description

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
feature	(IN)	the package name of the feature to test
version	(IN)	the version number of the package name to test

importNode()

Description

Imports a node from another document to this document. The returned node has no parent; it is `NULL`. The source node is not altered or removed from the original document; this method creates a new copy of the source node. If deep is TRUE, recursively imports the subtree under node; if it's FALSE, imports only the node itself.

Additional information is copied as appropriate to the `nodeType`, attempting to mirror the behavior expected if a fragment of XML source was copied from one document to another, recognizing that two documents may have different DTDs. See DOM 2.0 spec for specific action taken for each node type.

Syntax

```
xmlnode *importNode( xmlctx *ctx,
                     xmlnode *import,
                     boolean deep);
```

Parameter	IN / OUT	Description
ctx	(IN)	XML context
import	(IN)	node to be imported

Parameter	IN / OUT	Description
deep	(IN)	recursively import subtree?

insertBefore()

Description

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 into which new node will be inserted
newChild	(IN)	new child node to insert
refChild	(IN)	reference node before which insertion occurs

insertData()

Description

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, 0 is first position
arg	(IN)	new string to insert

isStandalone()

Description

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()

Description

Validates a node against the DTD. Returns 0 on success, else a nonzero error code (which can be looked up in the message file). This function is provided for applications which construct their own documents using the API or Class Generator, or both.. 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()

Description

“Normalizes” an element, or merges adjacent TEXT nodes. Adjacent TEXT nodes don't happen during a normal parse, only when extra nodes are inserted using the DOM.

Syntax

```
void normalize( xmlctx *ctx,
                xmlnode *elem);
```

Parameter	IN / OUT	Description
ctx	(IN)	XML context
elem	(IN)	pointer to element node

numAttributes()

Description

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()

Description

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 nodes structure

prefixToURI()

Description

Returns the matching URI given a namespace prefix and a node. If the given node doesn't have a matching prefix, its parent is tried, then *its* parent, and so on, all the way to the root node. If the prefix is undefined, `NULL` is returned.

Syntax

```
oratext *prefixToURI( xmlnode *node,
                      oratext *prefix);
```

Parameter	IN / OUT	Description
node	(IN)	Starting node
prefix	(IN)	Prefix to match

removeAttribute()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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()

Description

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 first character)
count	(IN)	length of old substring
arg	(IN)	replacement text

saveString()

Description

Saves a character string in the XML memory pool. The memory will be freed after an xmlclean or xmlterm call.

Syntax

```
oratext *saveString( xmlctx *ctx,
                     oratext *str);
```

Parameter	IN / OUT	Description
ctx	(IN)	XML context
str	(IN)	string to save

saveString2()

Description

Allocates memory and saves the NULL-terminated Unicode string in the XML string pool. Note that a Unicode string is terminated with TWO NULL bytes, not just one! Strings saved this way cannot be freed individually since they are stored head-to-tail in a single pool, for maximum compactness. The memory is reused only when the entire pool is freed, after an `xmlclean()` or `xmlterm()` calls. Use saveString() for saving single or multibyte strings.

Syntax

```
ub2 *saveString2( xmlctx *ctx,
                  ub2 *ustr);
```

Parameter	IN / OUT	Description
xtx	(IN)	LPX context
ustr	(IN)	Pointer to a unicode string

setAttribute()

Description

Creates 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()

Description

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

setAttrValue()

Description

Sets an attribute's value.

Syntax

```
void setAttrValue( xmlnode *attr,
                   const oratext *data);
```

Parameter	IN / OUT	Description
attr	(IN)	attribute whose value must be set
data	(IN)	new value for attribute

setCharData()

Description

Sets a TEXT or CDATA node's value.

Syntax

```
void setCharData( xmlnode *node,
                  const oratext *data);
```

Parameter	IN / OUT	Description
node	(IN)	node whose data must be set
data	(IN)	new text for node

setNamedItem()

Description

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 add node to
node	(IN)	new node to add
old	(IN)	pointer to replaced node

setNodeValue()

Description

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()

Description

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()

Description

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()

Description

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)	offset of start of substring
count	(IN)	length of substring

Namespace APIs

Description of 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 reuse 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.

Functions of Namespace APIs

Table 13–8 Summary of Functions of Namespace APIs

Function	Description
getAttrLocal() on page 13-84	Returns attribute local name.
getAttrNamespace() on page 13-84	Returns attribute namespace (URI).
getAttrPrefix() on page 13-85	Returns attribute prefix.
getAttrQualifiedName() on page 13-85	Returns attribute fully qualified name.
getNodeLocal() on page 13-85	Returns node local name.
getNodeNamespace() on page 13-86	Returns node namespace (URI).
getNodePrefix() on page 13-86	Returns node prefix.
getNodeQualifiedName() on page 13-86	Returns node qualified name.

getAttrLocal()

Description

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()

Description

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()

Description

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()

Description

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()

Description

This function returns the local name of this node.

Syntax

```
const oratext *getNodeLocal( const xmlnode *node);
```

Parameter	IN / OUT	Description
node	(IN)	node from which local name is retrieved

getNodeNamespace()

Description

Returns namespace for this node.

Syntax

```
const oratext *getNodeNamespace( const xmlnode *node);
```

Parameter	IN / OUT	Description
node	(IN)	node from which namespace is retrieved

getNodePrefix()

Description

Returns prefix for this node.

Syntax

```
const oratext *getNodePrefix( const xmlnode *node);
```

Parameter	IN / OUT	Description
node	(IN)	node from which prefix is retrieved

getNodeQualifiedName()

Description

Returns fully qualified name for this node.

Syntax

```
const oratext *getNodeQualifiedName( const xmlnode *node);
```

Parameter	IN / OUT	Description
node	(IN)	node from which name is retrieved

14

XSLT Processor for C

The XSL language processor is used to read XML documents and transform them into other XML documents with different styles. The C implementation of the XSL processor follows the XSL Transformations standard (version 1.0, November 16, 1999) and implements the required behavior of an XSL processor as specified in the XSLT specification.

This chapter describes the following sections:

- Data Structures and Types of XSLT APIs

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*

Data Structures and Types of XSLT APIs

Data Structures

Table 14–1 Summary of Data Types

Data Type	Definition	Description
oratext	typedef unsigned char oratext;	String pointer used for all data encodings, cast as needed; for UTF-16, to `(ub *)`.
ub4	typedef unsigned int ub4;	32-bit, or larger, unsigned integer.
uword	typedef unsigned int uword;	Native unsigned integer.
xmlctx	typedef struct xmlctx xmlctx;	Top-level XML context.
xmlnode	typedef struct xmlnode xmlnode;	Document node.

Table 14–1 Summary of Data Types (Cont.)

Data 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);
	sword (*comment)(void *ctx, const oratext *data);
	sword (*elementDecl)(void *ctx, const oratext *name,
	const oratext *content);
	sword (*attributeDecl)(void *ctx, const oratext *elem,
	const oratext *attr, const oratext *body);
	/* The following 5 fields are reserved for future use. */
	void (*empty1)();
	void (*empty2)();
	void (*empty3)();
	void (*empty4)();
	void (*empty5)();
	};
	typedef struct xmlsaxcb xmlsaxcb;
		SAX callback structure; SAX only.

Table 14–1 Summary of Data Types (Cont.)

Data Type	Definition	Description

xmlstream	`struct xmlstream;`
	`    typedef struct xmlstream xmlstream;`
	
	`    #define XMLSTREAM_OPENF(func)  uword`
	`            func(xmlstream *stream,  ub4 rsvd)`
	`    #define XMLSTREAM_WRITEF(func) uword`
	`            func(xmlstream *stream, ub4 rsvd)`
	`    #define XMLSTREAM_CLOSEF(func) uword`
	`            func(xmlstream *stream, ub4 rsvd)`
	
	`    struct xmlstream {`
	`      XMLSTREAM_OPENF((*open));`
	`      XMLSTREAM_WRITEF((*write));`
	`      XMLSTREAM_CLOSEF((*close));`
	
	`    /* User can associate any data to lpxstream structure`
	`      using this member so that he can access that inside`
	`      the callbacks which will always have xmlstream as`
	`      the firstargument. */`
	
	`    void *userdata;`
	`    };`
		Stream-based output.

xpnset	`typedef struct xpnset xpnset;`	XPath node set.
xpnsetele	`typedef struct xpnsetele xpnsetele;`	XPath node set element.
xpobj	`typedef struct xpobj xpobj;`	XPath object.

xpobjtyp	`typedef enum xpobjtyp (`
	`    XPOBJTYP_BOOL,       /* boolean */`
	`    XPOBJTYP_NUM,        /* number */`
	`    XPOBJTYP_STR,        /* string */`
	`    XPOBJTYP_NSET,       /* node set */`
	`    XPOBJTYP_RTFRAG    /* restult tree fragment */`
	`} xpobjtyp;`
		XPath object type.

Table 14–1 Summary of Data Types (Cont.)

Data Type	Definition	Description
xslctx	typedef struct xslctx xslctx;	Top-level XSL context.

xsloutputmethod	typedef enum {
	XSLMETHOD_UNKNOWN = 0,
	XSLMETHOD_TEXT,
	XSLMETHOD_HTML,
	} xsloutputmethod;
		Output method for XSL processing.

Functions of XSLT API

Table 14–2 Summary of XSLT Functions

Function	Description
xslprocess() on page 14-7	Processes an XSL Stylesheet with an XML document source.
xslprocessex() on page 14-7	Processes an XSL Stylesheet with an XML document source.
xslprocessxml() on page 14-8	Processes an XSL Stylesheet with an XML document source.
xslprocessdocfrag() on page 14-9	Processes an XSL Stylesheet with an XML document source.
xslinit() on page 14-10	Creates and initializes an XSL context.
xslgetbaseuri() on page 14-11	Retrieves baseURI associated with XSL context.
xslgetoutputdomctx() on page 14-11	Retrieves output DOM associated with the XSL context.
xslgetoutputsax() on page 14-11	Retrieves SAX object associated with the XSL context.
xslgetoutputstream() on page 14-12	Retrieves output stream associated with the XSL context.
xslgetresultdocfrag() on page 14-12	Retrieves resultant doc fragment associated with the XSL context.
xslgetxslctx() on page 14-12	Retrieves XML context for XML stylesheet associated with XSL context.
xslsettextparam() on page 14-13	Sets a toplevel parameter (variable).
xslgettextparam() on page 14-13	Returns a top level parameter(variable) value.
xslsetoutputdomctx() on page 14-14	Sets XML context to store processing result (as DOM).
xslsetoutputencoding() on page 14-15	Sets XSLT output encoding.

Table 14–2 Summary of XSLT Functions (Cont.)

Function	Description
xslsetoutputmethod() on page 14-15	Sets XSLT output method.
xslsetoutputsax() on page 14-16	Sets the SAX based events to be generated for the output.
xslsetoutputsaxctx() on page 14-16	Sets the SAX based events to be generated for the output.
xslsetoutputstream() on page 14-17	Sets the stream object specifying callbacks to generate the text stream based output.
xslresetallparams() on page 14-17	Resets all the top level parameters added.
xslterm() on page 14-18	Terminates an XSLT context object.

Table 14–3 Summary of XPath Functions

Function	Description
xpmakexpathctx() on page 14-18	Creates an XPath context.
xpfreexpathctx() on page 14-19	Parses an XPath expression.
xpparsexpathexpr() on page 14-19	Evaluates an XPath expression.
xpevalxpathexpr() on page 14-20	Retrieves values of evaluated xpath expression result.
xpgetxpobjtyp() on page 14-20	Retrieves values of evaluated xpath expression result.
xpgetbooleanval() on page 14-21	Retrieves values of evaluated xpath expression result.
xpgetnumval() on page 14-21	Retrieves values of evaluated xpath expression result.
xpgetstrval() on page 14-22	Retrieves values of evaluated xpath expression result.
xpgetnsetval() on page 14-22	Retrieves values of evaluated xpath expression result.
xpgetrtfragval() on page 14-22	Retrieves values of evaluated xpath expression result.
xpgetfirstnsetelem() on page 14-23	Retrieves values of evaluated xpath expression result.
xpgetnextnsetelem() on page 14-23	Retrieves values of evaluated xpath expression result.
xpgetnsetelemnode() on page 14-23	Retrieves values of evaluated xpath expression result.

xslprocess()

Description

Processes an XSL Stylesheet with an XML document source. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL.`

Syntax

```
uword xslprocess( xmlctx *docctx,
                  xmlctx *xslctx,
                  xmlctx *resctx,
                  xmlnode **result);
```

Argument	IN / OUT	Description
docctx	(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 code.

xslprocessex()

Description

Processes an XSL Stylesheet with an XML document source. User can specify a list of toplevel parameters to be set before parsing begins. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL.`

Syntax

```
uword xslprocessex( xmlctx *docctx,
                    xmlctx *sctx,
                    xmlctx *resctx,
                    size-t nparams,
                    oratext *params[],
                    oratext *vals[],
                    oratext *baseuri,
                    xmlnode **result);
```

Argument	IN / OUT	Description
docctx	(IN/OUT)	The XML document context
sctx	(IN)	The XSL stylesheet context
resctx	(IN)	The result document fragment context
nparams	(IN)	Number of (params, vals) pairs to be passed
params	(IN)	Array of name of pars (must be nparams in count)
vals	(IN)	Array of value of pars (must be nparams in count)
baseuri	(IN)	String giving the base uri for style sheet
result	(IN/OUT)	The result document fragment code

xslprocessxml()

Description

Processes an XSL Stylesheet with an XML document source. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL.`

This XSLT API are supposed to be used after the user:

- has created the XSLT processing context using `xslinit()`,
- has set the base URI using `xslinit()`,
- has set all the top level parameters using `xslsettextparam()`,
- has selected an output method using `xslsetoutput()`

The function will process the given XML file (specified by `docctx`) according to the XSL stylesheet, parameters, and output method. This call returns one of the following depending on the output scheme that the user selected:

- **Stream based output**: The output text has been redirected using the stream callbacks.
- **SAX based output**: User should already have all the SAX events sent by XSLT.
- **DOM based output**: User can make a call to the `xslgetresultdocgrag()` function to access the final Document Fragment created in context of the `resctx()` set by a call to `xslsetoutputdomctx()`.

Syntax

```
uword xslprocessxml( xslctx *xslSSctx,
                     xmlctx *docctx,
                     boolean normalize,
                     ub4 resvd);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context
docctx	(IN/OUT)	The XML document context
normalize	(IN)	If `TRUE`, normalize the input DOM tree
resvd		Reserved; should be set to `NULL`

xslprocessdocfrag()

Description

Processes an XSL Stylesheet with an XML document source. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL.`

This XSLT API are supposed to be used after the user:

- has created the XSLT processing context using `xslinit()`,
- has set the base URI using `xslinit()`,
- has set all the top level parameters using `xslsettextparam()`,
- has selected an output method using `xslsetoutput()`

The function will process the given XML file (specified by `docctx`) according to the XSL stylesheet, parameters, and output method. This call returns one of the following depending on the output scheme that the user selected:

- **Stream based output:** The output text has been redirected using the stream callbacks.
- **SAX based output:** User should already have all the SAX events sent by XSLT.
- **DOM based output:** User can make a call to the `xslgetresultdocgrag()` function to access the final Document Fragment created in context of the `resctx()` set by a call to `xslsetoutputdomctx()`.

Syntax

```
uword xslprocessxmldocfrag( xslctx *xslSSctx,
                            xmlctx *docctx,
                            xmlnode *docFrag,
                            boolean normalize,
                            ub4 resvd);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context.
docctx	(IN/OUT)	The XML document context.
docFrag	(IN)	The input document gragment node.
normalize	(IN)	If `TRUE`, normalize the input DOM tree.
resvd		Reserved; should be set to `NULL`

xslinit()

Description

Creates and initializes an XSL context. Returns the pointer to the XSL context object if successful; otherwise `NULL`.

Syntax

```
xslctx* xslinit ( uword *err,
                  xmlctx *xslctx,
                  const oratext *baseURI,
                  ub4 resvd);
```

Argument	IN / OUT	Description
err	(OUT)	`LPXERR_OK` on success, otherwise error code.
xslctx	(IN)	The XSL context for the input stylesheet.
baseURI	(IN)	The base URI for processing `include` and `import` statements.
resvd		Reserved for future use; should be set to `NULL.`

xslgetbaseuri()

Description

Retrieves baseURI associated with XSL context. Returns the pointer to the base URI if successful; otherwise `NULL`.

Syntax

```
const oratext* xslgetbaseuri ( xslctx *xslSSctx);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context to be queried.

xslgetoutputdomctx()

Description

Retrieves output DOM associated with the XSL context. Returns the pointer to the output DOM context object if successful; otherwise `NULL`.

Syntax

```
xmlctx* xslgetoutputdomctx( xslctx *xslSSctx);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context to be queried

xslgetoutputsax()

Description

Retrieves SAX object associated with the XSL context. Returns the pointer to the output SAX object if successful; otherwise `NULL`.

Syntax

```
xmlsaxcb* xslgetoutputsax( xslctx *xslSSctx);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context to be queried

xslgetoutputstream()

Description

Retrieves output stream associated with the XSL context. Returns the pointer to the output stream object if successful; otherwise `NULL`.

Syntax

```
xmlstream* xslgetoutputstream( xslctx *xslSSctx);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context to be queried.

xslgetresultdocfrag()

Description

Retrieves resultant doc fragment associated with the XSL context. Returns the pointer to the document fragment node if successful; otherwise `NULL`.

Syntax

```
xmlnode*  xslgetresultdocfrag( xslctx *xslSSctx);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context to be queried.

xslgetxslctx()

Description

Retrieves XML context for XML stylesheet associated with XSL context. Returns the pointer to the XML context object of the XSL stylesheet if successful; otherwise `NULL`.

Syntax

```
xmlctx* xslgetxslctx( xslctx *xslSSctx);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context to be queried

xslsettextparam()

Description

Sets a toplevel parameter (variable). Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL`.

Syntax

```
uword xslsettextparam ( xslctx *xslSSctx,
                        const oratext *param_name,
                        const oratext *param_value);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context to be queried.
param_name	(IN)	The name of parameter (variable) whose value is to be set.
param_value	(IN)	The parameter value.

xslgettextparam()

Description

Returns a top level parameter(variable) value. Returns the pointer to the variable / parameter value if successful, otherwise `NULL`.

Syntax

```
const oratext *xslgettextparam( xslctx *xslSSctx,
                                const oratext *param_name);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context.
param_name	(IN)	The name of parameter (variable) whose value is to be retrieved.

xslsetoutputdomctx()

Description

Sets XML context to store processing result (as DOM). Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL`.

A call to `xslsetoutputdomctx() overrides` any previous output method set using any previous calls to `xslsetoutputdomctx()`, `xslsetoutputstream()` or `xslsetoutputsax()`. Any subsequent processing of the XML input will be stored in form of a Document Fragment in the `xmlctx` specified by `resctx`.

`xslgetoutputdomctx()` can be used to retrieve the presently set `resctx`. If no DOM context is set or a call to `xslsetoutputdomctx()` has been overridden by a

subsequent call to `xslsetoutputstream()` or `xslsetoutputsax()`, then `xslgetoutputdomctx()` will return a `NULL` context indicating that the present output mode is not a DOM output mode.

Syntax

```
uword xslsetoutputdomctx( xslctx *xslSSctx,
                          xmlctx *resctx);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context
resctx	(IN)	The XML context object in which the output document fragment will be stored. This pointer must remain valid throughout, since this context is used to generate the processing event during subsequent processing of XML trees.

xslsetoutputencoding()

Description

Sets XSLT output encoding. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL`. Overrides the XSL Stylesheet settings.

This call is used to set an output method equivalent to XSL processing directive `xsl:output encoding ="..."`. If the XSL stylesheet contains one or more similar statements, then this function will override the effect of any previously present statements in XSL stylesheet.

Syntax

```
uword xslsetoutputencoding( xslctx *xslSSctx,
                            oratext* outcoding);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context
outcoding	(IN)	Encoding name for the output encoding

xslsetoutputmethod()

Description

Sets XSLT output method. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL`. Overrides the XSL Stylesheet settings.

This call is used to set an output method equivalent to XSL processing directive `xsl:output method = "text/xml/html"`. If the XSL stylesheet contains one or more similar statements, then this function will override the effect of any previously present statements in XSL stylesheet.

Syntax

```
uword xslsetoutputmethod( xslctx *xslSSctx,
                          xsloutputmethod method,
                          ub4 resvd);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context
method	(IN)	Output method to be selected
resvd		Reserved for future; must set to `NULL`

xslsetoutputsax()

Description

Sets the SAX based events to be generated for the output. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL`.

The user can make a call to `xslsetoutputsazctx()` to set the SAX context to be used by the SAX callbacks. If none is set, `NULL` will be passed to the SAX callbacks as SAX context. **Note:** just setting the callbacks is sufficient to invoke the SAX callbacks.

Syntax

```
uword xslsetoutputsax( xslctx *xslSSctx,
                       xmlsaxcb *s);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context
s	(IN)	The SAX callbacks to be used to generate the SAX events

xslsetoutputsaxctx()

Description

Sets the SAX based events to be generated for the output. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL`.

The user can make a call to `xslsetoutputsazctx()` to set the SAX context to be used by the SAX callbacks. If none is set, `NULL` will be passed to the SAX callbacks as SAX context. **Note:** just setting the callbacks is sufficient to invoke the SAX callbacks.

Syntax

```
uword xslsetoutputsaxctx( xslctx *xslSSctx,
                          void *saxctx);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context
saxctx	(IN)	the SAX context

xslsetoutputstream()

Description

Sets the stream object specifying callbacks to generate the text stream based output. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL`.

Syntax

```
uword xslsetoutputstream( xslctx *xslSSctx,
                          xmlstream *stream);
```

Argument	IN / OUT	Description
stream	(IN)	The stream structure giving the callbacks to be used.
xslSSctx	(IN)	The XSL processing context

xslresetallparams()

Description

Resets all the top level parameters added. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL.`

Syntax

```
uword xslresetallparams( xslctx *xslSSctx);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context

xslterm()

Description

Terminates an XSLT context object. Returns `LPXERR_OK` on success, otherwise `LPXERR_FAIL`.

Syntax

```
uword xslterm( xslctx *xslSSctx);
```

Argument	IN / OUT	Description
xslSSctx	(IN)	The XSL processing context.

xpmakexpathctx()

Description

Creates an XPath context. Returns an XPath context object; this call never fails.

Syntax

```
xpctx *xpmakexpathctx( xmlctx *ctx,
                       xmlnode *xslnode,
                       xmlnode* xml_node,
                       oratext* baseURI,
                       size-t nctxels,
                       xmlnode ** ctxnodes);
```

Argument	IN / OUT	Description
sctx	(IN)	XSL context; could be `NULL`.

Argument	IN / OUT	Description
xslnode	(IN)	The XSL nodeused for namespace expansion; could be NULL.
xml_node	(IN)	The context node; set to NULL for parsing.
baseURI	(IN)	The base URI for parsing.
nctxels	(IN)	The number of nodes in the current node set.
ctxnodes	(IN)	The current node set.

xpfreexpathctx()

Description

Frees an XPath context.

Syntax

```
void xpfreexpathctx( xpctx *xctx);
```

Argument	IN / OUT	Description
xctx	(IN)	The XPath context to be freed.

xpparsexpathexpr()

Description

Parses an XPath expression. Returns the expression tree on success; otherwise, NULL.

Syntax

```
xpexpr *xpparsexpathexpr( xpctx *xctx,
                        oratext *expr,
                        sword *err);
```

Argument	IN / OUT	Description
xctx	(IN/OUT)	The XPath context
expr	(IN)	The expression in the form of a string

Argument	IN / OUT	Description
err	(OUT)	The error code

xpevalxpathexpr()

Description

Evaluates an XPath expression. Returns the result object on success; otherwise, `NULL.` The type of the result object, obtained using `xpgetxpobjtyp()`, is one of the following:

- `XPOBJTYP_BOOL`
- `XPOBJTYP_NUM`
- `XPOBJTYP_STR`
- `XPOBJTYP_NSET`
- `XPOBJTYP_RTFRAG`

Syntax

```
xpobj *xpevalxpathexpr( xpctx *xctx,
                       xpexpr *exprtree,
                       sword *err);
```

Argument	IN / OUT	Description
xctx	(IN/OUT)	The XPath context
exprtree	(IN)	The expression in the form of a tree
err	(OUT)	The error code

xpgetxpobjtyp()

Description

Retrieves values of evaluated xpath expression result. Returns the expression value on success; otherwise, `NULL.`

Syntax

```
xpobjtyp xpgetxpobjtyp( xpobj *xobj);
```

Argument	IN / OUT	Description
xobj	(IN)	The expression object

xpgetbooleanval()

Description

Retrieves values of evaluated xpath expression result. Returns the expression value on success; otherwise, `NULL`.

Syntax

```
boolean xpgetbooleanval (xpobj *xobj);
```

Argument	IN / OUT	Description
xobj	(IN)	The expression object

xpgetnumval()

Description

Retrieves values of evaluated xpath expression result. Returns the expression value on success; otherwise, `NULL`.

Syntax

```
double xpgetnumval( xpobj *xobj);
```

Argument	IN / OUT	Description
xobj	(IN)	The expression object

xpgetstrval()

Description

Retrieves values of evaluated xpath expression result. Returns the expression value on success; otherwise, `NULL`.

Syntax

```
oratext* xpgetstrval( xpobj *xobj);
```

Argument	IN / OUT	Description
xobj	(IN)	The expression object.

xpgetnsetval()

Description

Retrieves values of evaluated xpath expression result. Returns the expression value on success; otherwise, `NULL`.

Syntax

```
xpnset* xpgetnsetval( xpobj *xobj);
```

Argument	IN / OUT	Description
xobj	(IN)	The expression object.

xpgetrtfragval()

Description

Retrieves values of evaluated xpath expression result. Returns the expression value on success; otherwise, `NULL`.

Syntax

```
xmlnode*  xpgetrtfragval(xpobj *xobj);
```

Argument	IN / OUT	Description
xobj	(IN)	The expression object

xpgetfirstnsetelem()

Description

Retrieves values of evaluated xpath expression result. Returns the expression value on success; otherwise, `NULL`.

Syntax

```
xpnsetele* xpgetfirstnsetelem( xpnset *nset);
```

Argument	IN / OUT	Description
nset	(IN)	The node set

xpgetnextnsetelem()

Description

Retrieves values of evaluated xpath expression result.   Returns the expression value on success; otherwise, `NULL`.

Syntax

```
xpnsetele* xpgetnextnsetelem( xpnsetele *nsetele);
```

Argument	IN / OUT	Description
xpnstele*	(IN)	The node set element.

xpgetnsetelemnode()

Description

Retrieves values of evaluated xpath expression result. Returns the expression value on success; otherwise, `NULL`.

Syntax

```
xmlnode *xpgetnsetelemnode( xpnsetele *nsetele);
```

Argument	IN / OUT	Description
xpnstele*	(IN)	The node set element

15 XML Schema Processor for C

XML Schema is an alternative to XML DTDs. It defines the document organization, like DTD, but it also provides data typing of element content and additional features. XML Schema provides many built-in datatypes such as scalar, real, date and time, URIs, and encoded binary data. User-defined datatypes can be build from these primitive datatypes through extension or restriction.

XML Schema is still beta. This version implements the May 2001 W3C Recommendation. This beta implementation is incomplete at present; not all schema features are implemented.

This chapter describes the following section:

- Calling Sequence of XML Schema Processor for C
- Data Types of XML Schema Processor for C
- Functions and Methods of XML Schema Processor for C

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*

XML Schema Processor for C

Calling Sequence of XML Schema Processor for C

The sequence of calls to the processor is `schemaInitialize()`, `schemaValidate()`, ... , `schemaValidate()`, `schemaTerminate()`. The `initialize()` call is invoked once at the beginning of a session; it returns a Schema context that is used throughout the session.

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 nonzero 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.

The instance document to be validated is first parsed with the XML parser. The root element for the instance is then passed to the Schema validate function, along with an optional Schema URL. If optimal schema is provided, it will be loaded and become the default schema. More documents may then be processed using the same Schema context. When the session is over, the `schemaTerminate()` function is called, which releases all memory allocated by the loaded Schemas.

Data Types of XML Schema Processor for C

Table 15–1 Data Types of XML Schema Processor for C

Data Type	Definition	Description
xsd	typedef struct xsd xsd;	Schema structure.
xsdctx	typedef struct xsdctx xsdctx;	Schema processor context.

Note: The contents of `xsdctx` and `xsd` are private (opaque) and must not be accessed by users.

Functions and Methods of XML Schema Processor for C

Table 15–2 Summary of Functions and Methods of XML Schema Processor for C

Function / Method	Description
schemaInitialize() on page 15-3	Initialize the schema processor
schemaLoad() on page 15-3	Load an XML Schema file offline
schemaTarget() on page 15-4	Return target namespace URI for schema
schemaTerminate() on page 15-5	Shut down the Schema processor
schemaValidate() on page 15-5	Validate an instance document against a Schema

schemaInitialize()

Description

Initializes the XML Schema processor; must be called before any Schema validation can take place. The same context may be used repeatedly for validating multiple documents. Returns a pointer to the allocated context; otherwise returns `NULL`.

Syntax

```
xsdctx *schemaInitialize( xmlctx *ctx,
                          uword *err);
```

Arguments

Argument	IN / OUT	Description
ctx	(IN)	XML parser context used to allocate memory of the Schema context.

schemaLoad()

Description

Loads Schema and returns the error code, or `0` for success.

To indicate that "nsp" is not provided, specify `NULL`; to indicate that it's not in any target namespace, specify an empty string, ""; otherwise, provide a namespace URI such as “http://www.example.com/Report”. If “nsp” is provided, then the following must be all true:

- If there is a namespace, then its actual value must be identical to the actual value of the `targetNamespace` [attribute] of imported schema.
- If there is no namespace, then imported schema must have no `targetNamespace` [attribute].

Syntax

```
uword schemaLoad( xsdctx *scctx,
                  oratext *uri,
                  oratext *nsp,
                  xsd **schema);
```

Argument	IN / OUT	Description
scctx	(IN)	XML context used to allocate memory for the Schema context
uri	(IN)	URL of schema, in compiler character set
nsp	(IN)	Namespace of schema; optional; in compiler character set
schema	(OUT)	Returned schema structure

schemaTarget()

Description

Returns target namespace URI for schema.

Syntax

```
oratext *schemaTarget( xsd *schema);
```

Argument	IN / OUT	Description
xsd	(IN)	Schema structure

schemaTerminate()

Description

Tears down the Schema processor, releasing all allocated memory. The context may not be used again.

Syntax

```
void schemaTerminate( xsdctx *scctx);
```

Argument	IN / OUT	Description
scctx	(IN)	Schema context to terminate.

schemaValidate()

Description

Validates an instance document against a Schema. If the default Schema is provided, the named schema is loaded and will become the default. Returns `0` on success (the document is structurally valid and all datatypes check out), or an error code on failure.

Syntax

```
uword schemaValidate( xsdctx *scctx,
                     xmlnode *root,
                     xsd *schema);
```

Argument	IN / OUT	Description
ctx	(IN)	Schema context.
root	(IN)	Root element of instance of document to validate.
schema	(IN)	URI of default Schema, in compiler character set.

Part III

XDK for C++ Packages

This section contains the following chapters:

16 XML Parser for C++

This chapter contains the following sections:

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*

Attr Class

Description of Attr

This class contains methods for accessing the name and value of a single document node attribute.

Methods of Attr

Table 16–1 Summary of Methods of Attr

Method	Description
getName() on page 16-2	Return name of attribute
getValue() on page 16-2	Return “value” (definition) of attribute
getSpecified() on page 16-3	Return attribute's “specified” flag value
setValue() on page 16-3	Set an attribute's value

getName()

Description

Returns name of attribute

Syntax

```
DOMString getName();
```

getValue()

Description

Return “value” (definition) of attribute

Syntax

```
DOMString getValue();
```

getSpecified()

Description

Returns value of attribute's “specified” flag. From the DOM. 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()

Description

Sets an attribute's “value”

Syntax

```
void setValue(DOMString value);
```

Parameter	Description
value	Attribute’s new value

CDATASection Class

Description of CDATASection

This class implements the CDATA node type, a subclass of Text. It inherits all of its methods from the Text Class.

Comment Class

Description of Comment

This class implements the COMMENT node type, a subclass of CharacterData. It inherits all of this methods from the CharacterData Class.

CharacterData Class

Description of CharacterData

This class contains methods for accessing and modifying the data associated with text nodes.

Methods of CharacterData

Table 16–2 Summary of Methods of CharacterData

Method	Description
appendData() on page 16-5	Appends a string to this node's data.
deleteData() on page 16-5	Deletes a substring to this text node's data.
getData() on page 16-5	Retrieves value of a text node.
getLength() on page 16-6	Returns length of a text node's data.
insertData() on page 16-6	Inserts a string into this node's data.
replaceData() on page 16-6	Replaces a substring in this node's data.
setData() on page 16-7	Sets value of a text node.
substringData() on page 16-7	Fetches a substring of this node's data.

appendData()

Description

Appends a string to this node’s data.

Syntax

```
void appendData(DOMString arg);
```

Parameter	Description
arg	String to append

deleteData()

Description

Deletes a substring to this text node’s data.

Syntax

```
void deleteData(  unsigned long offset,
                  unsigned long count);
```

Parameter	Description
count	Number of characters to remove
offset	Start of substring to remove (`0` is the first char)

getData()

Description

Retrieves data (value) of a text node as a DOMString.

Syntax

```
DOMString getData();
```

getLength()

Description

Returns the length of a text node's data

Syntax

```
size_t getLength();
```

insertData()

Description

Inserts a string into this node's data

Syntax

```
void insertData(  unsigned long offset,
                  DOMString arg);
```

Parameter	Description
arg	String to insert.
offset	Insertion point (0 means before first character).

replaceData()

Description

Replaces a substring in this node's data.

Syntax

```
void replaceData(  unsigned long offset,
                    unsigned long count,
                      DOMString arg);
```

Parameter	Description
arg	string to insert
count	number of characters to replace

Parameter	Description
offset	insertion point (0 means before first char)

setData()

Description

Sets data, or value, of a text node.

Syntax

```
void setData(  DOMString data);
```

Parameter	Description
data	Data for node

substringData()

Description

Fetches a substring of a node's data.

Syntax

```
DOMString substringData(  unsigned long offset,
                         unsigned long count);
```

Parameter	Description
count	length of substring
offset	start of substring (0 means first char)

Document Class

Description of Document

This class contains methods for creating and retrieving nodes.

Methods of Document

Table 16–3 Summary of Methods of Document

Method	Description
createAttribute() on page 16-9	Creates an ATTRIBUTE node.
createAttributeNS() on page 16-9	Create san ATTRIBUTE node with namespace information.
createCDATASection() on page 16-10	Creates a CDATA node.
createComment() on page 16-10	Creates a COMMENT node.
createDocumentFragment() on page 16-10	Creates a DOCUMENT_FRAGMENT node.
createElement() on page 16-11	Create san ELEMENT node.
createElementNS() on page 16-11	Creates an ELEMENT node with namespace information.
createEntityReference() on page 16-11	Creates an ENTITY_REFERENCE node.
createProcessingInstruction() on page 16-12	Create a PROCESSING_INSTRUCTION node.
createTextNode() on page 16-12	Creates a TEXT node.
getElementByID() on page 16-12	Returns the element node which has the given ID.
getElementsByTagName() on page 16-13	Selects nodes based on tag name.
getImplementation() on page 16-14	Returns DTD for document.
importNode() on page 16-14	Copies a node from another document into this document.
saveString() on page 16-15	Allocates memory and saves the given string.

createAttribute()

Description

Creates a new attribute node, and returns a pointer to that node. Use `setValue()` to set its value.

Syntax

```
Attr* createAttribute(  DOMString name,
                       DOMString value);
```

Parameter	Description
name	name of attribute
value	value of attribute

createAttributeNS()

Description

Creates a new attribute node with namespace information, and returns a pointer to that node. Use `setValue()` to set its value.

Syntax

```
Attr* createAttribute( DOMString nspuri,
                       DOMString qname,
                       DOMSring value);
```

Parameter	Description
nspuri	element's namespace URI
qname	qualified name of attribute
value	value of attribute

createCDATASection()

Description

Creates a new CDATA node with the given contents, and returns a pointer to that node.

Syntax

```
CDATASection* createCDATASection(  DOMString data);
```

Parameter	Description
data	Contents of node.

createComment()

Description

Creates a new comment node with given contents, and returns a pointer to that node.

Syntax

```
Comment* createComment(  DOMString data);
```

Parameter	Description
data	contents of node

createDocumentFragment()

Description

Creates a new document fragment node and returns a pointer to that node.

Syntax

```
DocumentFragment* createDocumentFragment();
```

createElement()

Description

Creates a new element node with the given tag name and returns a pointer to that node.

Syntax

```
Element* createElement(  DOMString tagName);
```

Parameter	Description
tagName	element's tag name

createElementNS()

Description

Creates a new element node with the given (tag) name and namespace information, and returns a pointer to that node.

Syntax

```
Element* createElementNS( DOMString nspuri,
                          DOMSring qname);
```

Parameter	Description
nspuri	element's namespace URI
qname	element's qualified name

createEntityReference()

Description

Creates a new entity reference node and returns a pointer to that node.

Syntax

```
EntityReference* createEntityReference(  DOMString name);
```

Parameter	Description
name	name of entity to reference

createProcessingInstruction()

Description

Creates a new processing instruction node and returns a pointer to that node.

Syntax

```
ProcessingInstruction* createProcessingInstruction( DOMString target,
                                                    DOMString data);
```

Parameter	Description
data	data for node
target	target part of PI

createTextNode()

Description

Creates a new TEXT node and returns a pointer to that node.

Syntax

```
Text* createTextNode(  DOMString data);
```

Parameter	Description
data	data for node

getElementByID()

Description

Returns the element node with the given ID. If no such ID is defined (or other problems are found), returns `NULL`.

Syntax

```
Element* getElementByID(  DOMString name);
```

Parameter	Description
name	element ID

getElementsByTagName()

Description

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. If elem is `NULL`, the entire document is searched. The special value "*" matches all tags. If no matches are found, returns `NULL`. The options are described in the following table.

Syntax	Description
NodeList* getElementsByTagName(Element *elem, DOMString tagname);	Matches Element by tag name.
NodeList* getElementsByTagNameNS(Element *elem, DOMString nspuri, DOMString local);	Matches Element by tag name and namespace information.

Parameter	Description
elem	root node
tagname	tag name to select
nspuri	namespace URI
local	namespace local name

getImplementation()

Description

Returns the DOMImplementation structure. This function is currently not in use, and is reserved for future DOM implementations.

Syntax

```
DOMImplementation* getImplementation();
```

importNode()

Description

Imports a node from another document to this document, and returns a pointer to that node. The returned node has no parent; its value is NULL. The source node is not altered or removed from the original document; this method creates a new copy of the source node. If deep is TRUE, recursively imports the subtree under node; if it is FALSE, imports only the node itself.

Additional information is copied as appropriate to the nodeType, attempting to mirror the behavior expected if a fragment of XML source was copied from one document to another, recognizing that two documents may have different DTDs. See SOM 2.0 spec for specific action taken for each node type.

Syntax

```
xmlnode *importNode(  smlctx *ctx,
                      xmlnode *import,
                      boolean deep);
```

Parameter	Description
ctx	XML context.
deep	Recursively import subtree - TRUE or FALSE.
import	Node to be imported.

saveString()

Description

Allocates memory and saves the given string, returning a pointer to that string. Used to store locally generated strings.

Syntax

```
DOMString saveString(  DOMString str);
```

Parameter	Description
str	string to save

DocumentType Class

Description of DocumentType

This class contains methods for accessing information about the Document Type Definition (DTD) of a document.

Methods of DocumentType

Table 16–4 Summary of Methods of DocumentType

Method	Description
getName() on page 16-16	Returns name of DTD.
getEntities() on page 16-16	Returns NamedNodeMap of DTD's (general) entities.
getNotations() on page 16-16	Returns NamedNodeMap of DTD's notations.

getName()

Description

Returns name of DTD.

Syntax

```
DOMString getName();
```

getEntities()

Description

Returns map of DTD's (general) entities.

Syntax

```
NamedNodeMap* getEntities();
```

getNotations()

Description

Returns map of DTD's notations.

Syntax

```
NamedNodeMap* getNotations();
```

DOMImplementation Class

Description of DOMImplementation

This class contains methods relating to the specific DOM implementation supported by the parser.

Methods of DOMImplementation

Table 16–5 Summary of Methods of DOMImplementation

Method	Description
createDocument() on page 16-17	Creates and returns a `DOCUMENT` node.
createDocumentType() on page 16-18	Creates and returns a `DOCUMENT_TYPE` (DTD) node.
hasFeature() on page 16-18	Tests if the DOMImplementation implements a specific feature.

createDocument()

Description

Creates a `DOCUMENT` node and returns a pointer to that node. When DTD is not `NULL`, its `Node.ownerDocument` attribute is set to the document being created.

Syntax

```
Document *createDocument( DOMString uri,
                          DOMString qname,
                          DocumentType *dtd);
```

Parameter	Description
dtd	document type (DTD)
qname	qualified name of the new document element
uri	namespace URI of the new document element

createDocumentType()

Description

Creates a DOCUMENT_TYPE (DTD) node and returns a pointer to that node.

Syntax

```
DocumentType *createDocumentType( DOMString qname,
                                  DOMString pubid,
                                  DOMString sysid);
```

Parameter	Description
pubid	external subset system identifier
qname	qualified name of the new document element
sysid	external subset system identifier

hasFeature()

Description

Tests if the DOM implementation implements a specific feature. Returns TRUE if the feature is supported, FALSE otherwise.

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.

Element Class

Description of Element

This class contains methods pertaining to element nodes.

Methods of Element

Table 16–6 Summary of Methods of Element

Method	Description
getAttribute() on page 16-19	Selects an attribute given its name
getAttributeNode() on page 16-20	Removes an attribute given its name
getElementsByTagName() on page 16-20	Returns a list of element nodes with the given tag name
getElementsByTagNameNS() on page 16-20	Returns a list of element nodes with the given tag name, namespace aware
getTagName() on page 16-21	Returns the node's tag name
initialize() on page 16-21	Initializes a new element node
normalize() on page 16-22	Normalizes an element (merge adjacent text nodes)
removeAttribute() on page 16-22	Removes an attribute given its name
removeAttributeNode() on page 16-22	Removes an attribute node
setAttribute() on page 16-23	Creates a new attribute given its name and value
setAttributeNode() on page 16-23	Adds a new attribute node

getAttribute()

Description

Returns “value” (definition) of named attribute.

Syntax

```
DOMString getAttribute(  DOMString name);
```

Parameter	Description
name	name of attribute

getAttributeNode()

Description

Returns pointer to named attribute, or NONE if not found.

Syntax

```
Attr* getAttributeNode(  DOMString name);
```

Parameter	Description
name	name of attribute

getElementsByTagName()

Description

Creates and returns a list of matching elements.

Syntax

```
NodeList* getElementsByTagName( DOMString name);
```

Parameter	Description
name	tag name to match, "*" for all.

getElementsByTagNameNS()

Description

Creates and returns a list of matching elements, namespace aware.

Syntax

```
NodeList* getElementsByTagNameNS( DOMString nspuri,
                                  DOMString local);
```

Parameter	Description
local	Local name to match, “*” for all.
nspuri	Namespace URI.

getTagName()

Description

Returns the tag name of the element. 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 DOM Working Group considers it worthwhile to support both, given the different constituencies the DOM API must satisfy.

Syntax

```
DOMString getTagName();
```

initialize()

Description

Initializes a newly allocated element node. Returns `0` on success, or an error code on failure.

Syntax

```
uword initialize( Document *doc,
                  DOMString nspuri,
                  DOMString qname);
```

Parameter	Description
doc	XML document.
nspuri	Namespace URI.
qname	Qualified name.

normalize()

Description

Normalizes an element; merge all adjacent TEXT nodes.

Syntax

```
void normalize();
```

removeAttribute()

Description

Removes the named attribute.

Syntax

```
void removeAttribute(  DOMString name);
```

Parameter	Description
name	name of attribute to remove

removeAttributeNode()

Description

Removes the named attribute node and passes it back.

Syntax

```
Attr* removeAttributeNode(  Attr* oldAttr);
```

Parameter	Description
name	attribute to remove

setAttribute()

Description

Creates a new attribute. Returns a pointer to the newly created attribute.

Syntax

```
Attr* setAttribute( DOMString name,
                    DOMString value);
```

Parameter	Description
name	name of new attribute
value	value of new attribute

setAttributeNode()

Description

Sets (adds) new attribute node.

Syntax

```
boolean setAttributeNode( Attr* newAttr,
                          Attr** oldAttr);
```

Parameter	Description
newAttr	pointer to a new attribute
oldAttr	returned pointer to replaced attribute

Entity Class

Description of Entity

This class implements the `ENTITY` node type, a subclass of `Node`.

Methods of Entity

Table 16–7 Summary of Methods of Entity

Method	**Description**
getNotationName() on page 16-24	Returns entity's `NDATA` (notation name).
getPublicId() on page 16-24	Returns entity's public ID.
getSystemId() on page 16-24	Returns entity's system ID.

getNotationName()

Description

Returns an entity node's notation name (`NDATA`).

Syntax

```
DOMString* getNotationName();
```

getPublicId()

Description

Returns an entity node's public ID.

Syntax

```
DOMString getPublicId();
```

getSystemId()

Description

Returns an entity node's system ID.

Syntax

```
DOMString getSystemId();
```

EntityReference Class

Description of EntityReference

This class implements the `ENTITY_REFERENCE` node type, a subclass of `Node.`

NamedNodeMap Class

Desciption of NamedNodeMap

This class contains methods for accessing the number of nodes in a node map and fetching individual nodes.

Methods of NamedNodeMap

Table 16–8 Summary of Methods of NamedNodeMap

Method	Description
free() on page 16-25	Frees the named node map.
getLength() on page 16-26	Returns number of nodes in map.
getNamedItem() on page 16-26	Selects a node by name.
item() on page 16-26	Returns *n*th node in map.
removeNamedItem() on page 16-27	Removes the named node from map.
setNamedItem() on page 16-27	Sets a node into the map.

free()

Description

Frees the named node map.

Syntax

```
void free();
```

getLength()

Description

Returns number of nodes in map.

Syntax

```
size_t getLength();
```

getNamedItem()

Description

Selects the node with the given name from the map. Returns `NULL` if not found.

Syntax

```
Node* getNamedItem(  String name);
```

Parameter	Description
name	Name of node to select

item()

Description

Returns a pointer to the *n*th node in node map.

Syntax

```
Node* item(  size_t index);
```

Parameter	Description
size_t	index; zero-based node number

removeNamedItem()

Description

Removes the node with the given name from the node map. Returns a pointer to removed node, NULL if not found.

Syntax

```
Node* removeNamedItem(  String name);
```

Parameter	Description
name	name of node to remove

setNamedItem()

Description

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

Node Class

Description of Node

This class contains methods for details about a document node

Methods of Node

Table 16–9 Summary of Methods of Node

Method	Description
appendChild() on page 16-29	Appends a new child to the end of the current node’s list of children.
cloneNode() on page 16-29	Clones an existing node and optionally all its children.
getAttributes() on page 16-30	Returns structure contains all defined node attributes.
getChildNode() on page 16-30	Returns specific indexed child of given node.
getChildNodes() on page 16-31	Returns structure contains all child nodes of given node.
getContext() on page 16-31	Gets node’s context.
getFirstChild() on page 16-31	Returns first child of given node.
getLastChild() on page 16-31	Returns last child of given node.
getLocal() on page 16-31	Returns the local name of the node.
getName() on page 16-32	Returns name of node.
getNamespace() on page 16-32	Returns a node's namespace.
getNextSibling() on page 16-32	Returns a node's next sibling.
getOwnerDocument() on page 16-32	Returns document node which contains a node.
getParentNode() on page 16-33	Returns parent node of given node.
getPrefix() on page 33	Returns the namespace prefix for the node.
getPreviousSibling() on page 16-33	Returns the previous sibling of the current node.
getQualifiedName() on page 16-33	Returns namespace qualified node of given node.
getType() on page 16-34	Returns numeric type-code of node.

Table 16–9 Summary of Methods of Node (Cont.)

Method	Description
getValue() on page 16-34	Returns “value” (data) of node.
hasAttributes() on page 16-34	Determines if node has any defined attributes.
hasChildNodes() on page 16-35	Determines if node has children.
insertBefore() on page 16-35	Inserts new child node into a node's list of children.
numChildNodes() on page 16-35	Returns count of number of child nodes of given node.
print() on page 16-36	Formats XML document to stream or buffer.
printSize() on page 16-36	Determines size of formatted XML document (without outputting).
removeChild() on page 16-37	Removes a node from the current node's list of children.
replaceChild() on page 16-37	Replaces a child node with another.
setValue() on page 16-37	Sets a node's value (data).

appendChild()

Description

Appends a new child to the current node's list of children and returns a pointer to that node.

Syntax

```
Node* appendChild(  Node *newChild);
```

Parameter	Description
newChild	new child node

cloneNode()

Description

Returns a duplicate of this node; serves as a generic copy constructor for nodes. Returns a pointer to new clone. 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()

Description

Retrieves structure of all attributes for node. Returns a pointer to structure describing all attributes for node, or `NULL` if no attributes are defined

Syntax

```
NamedNodeMap* getAttributes();
```

getChildNode()

Description

Returns one of the node's children as a pointer to child of node found through the index.

Syntax

```
Node* getChildNode(  uword index);
```

Parameter	Description
index	child number, starting at 0

getChildNodes()

Description

Returns node's children as a pointer to structure describing all the node's children.

Syntax

```
NodeList* getChildNodes();
```

getContext()

Description

Returns the node's context

Syntax

```
xmlctx* getContext();
```

getFirstChild()

Description

Returns a pointer to the node's first child.

Syntax

```
Node* getFirstChild();
```

getLastChild()

Description

Returns a pointer to the node's last child.

Syntax

```
Node* getLastChild();
```

getLocal()

Description

Returns the node's local name

Syntax

```
DOMString getLocal();
```

getName()

Description

Returns name of node, or `NULL` if the node has no name.

Syntax

```
DOMString getName();
```

getNamespace()

Description

Returns the node's namespace; may be `NULL`.

Syntax

```
DOMString getNamespace();
```

getNextSibling()

Description

Returns the next sibling of the node; or the next child of the node’s parent. `NULL` returned if the current child is the last.

Syntax

```
Node* getNextSibling();
```

getOwnerDocument()

Description

Returns document node which contains the current node, as a pointer to that document node.

Syntax

```
Document* getOwnerDocument();
```

getParentNode()

Description
Returns node's parent.

Syntax
```
Node* getParentNode();
```

getPrefix()

Description
Returns the namespace prefix of node; may be `NULL`.

Syntax
```
DOMString getPrefix();
```

getPreviousSibling()

Description
Returns the previous sibling of the node, or the previous child of the node’s parent. `NULL` returned if the current child is the first.

Syntax
```
Node* getPreviousSibling();
```

getQualifiedName()

Description
Returns the fully qualified (namespace) name of node.

Syntax
```
DOMString getQualifiedName();
```

getType()

Description

Return numeric type code for the node. The possible codes are:

DOCUMENT_FRAGMENT_NODE	ENTITY_NODE
ELEMENT_NODE	PROCESSING_INSTRUCTION_NODE
NOTATION_NODE	COMMENT_NODE
ATTRIBUTE_NODE	DOCUMENT_NODE
TEXT_NODE	DOCUMENT_TYPE_NODE
ENTITY_REFERENCE_NODE	CDATA_SECTION_NODE

Syntax

```
short getType();
```

getValue()

Description

Returns “value” (data) of node, or `NULL` if the node has no value.

Syntax

```
DOMString getValue();
```

hasAttributes()

Description

Determines if node has any defined attributes. Returns `TRUE` if node has attributes, `FALSE` otherwise.

Syntax

```
boolean hasAttributes();
```

hasChildNodes()

Description

Determines if node has any children; returns `TRUE` if node has children, `FALSE` otherwise.

Syntax

```
boolean hasChildNodes();
```

insertBefore()

Description

Inserts a new child node into the list of children of a parent, before the reference node; the pointer to the new child passed back. If the reference node passed in `NULL`, appends the new node to the end.

Syntax

```
Node* insertBefore( Node *newChild,
                    Node *refChild);
```

Parameter	Description
newChild	new node to insert
refChild	reference node; new node comes before

numChildNodes()

Description

Returns count of node's children; may be `0`.

Syntax

```
uword numChildNodes();
```

print()

Description

Formats XML document to stream or buffer. The options are described in the following table.

Syntax	Description
void print(File *out = stdout, uword level = 0, uwoard step = 4);	Creates a new parser object.
void print(DOMString buffer, size_t buffsize, uword level = 0, uword step =4);	Creates a new parser object with a given id.

Parameter	Description
out	stdio output stream
level	starting indentation level
step	spaces for each indentation level
buffer	destination buffer
bufsize	size of destination buffer

printSize()

Description

Returns size (in bytes) of formatted XML document, without actually outputting it. Useful for determining final size of document for buffer allocation purposes. Caution, this call is almost as expensive as really outputting the document.

Syntax

```
size_t printSize( uword level = 0,
                  uwoard step = 4);
```

Parameter	Description
level	starting indentation level
step	spaces for each indentation level

removeChild()

Description

Removes a child node from the current node's list of children, and passes back the removed child.

Syntax

```
Node* removeChild();
```

replaceChild()

Description

Replaces one node with another. `newChild` replaces `oldChild` in the list of children in `oldChild`'s parent. `oldChild` is passed back.

Syntax

```
Node* replaceChild( Node *newChild,
                    Node *oldChild);
```

Parameter	Description
newChild	new replacement node
oldChild	old node being replaced

setValue()

Description

Sets a node's “value” (data).

Syntax

```
void setValue(DOMString data);
```

Parameter	Description
data	new data for the node

NodeList Class

Description of NodeList

This class contains methods for extracting nodes from a `NodeList`.

Methods of NodeList

Table 16–10 Summary of Methods of NodeList

Method	Description
free() on page 16-39	Returns number of nodes in list.
getLength() on page 16-39	Returns *n*th node in list.
item() on page 16-39	Frees the node list.

free()

Description

Frees the node list.

Syntax

```
void free();
```

getLength()

Description

Returns number of nodes in list.

Syntax

```
size_t getLength();
```

item()

Description

Returns *n*th node in node list.

Syntax

```
Node* item( size_t index);
```

Parameter	Description
index	0-based node number.

Notation Class

Description of Notation

This class implements the NOTATION node type, a subclass of **Node**.

Methods of Notation

Table 16–11 Summary of Methods of Notation

Method	Description
getData() on page 16-41	Returns notation’s data.
getTarget() on page 16-41	Returns notation’s target.
setData() on page 16-41	Sets notation’s data.

getData()

Description

Returns a notation's data.

Syntax

```
DOMString getData();
```

getTarget()

Description

Returns a notation's target.

Syntax

```
DOMString getTarget();
```

setData()

Description

Sets a notation's data.

Syntax

```
void setData(  DOMString data);
```

Parameter	Description
data	new data

ProcessingInstruction Class

Description of ProcessingInstruction

This class implements a subclass of `Node`, `PROCESSING_INSTRUCTION` node type.

Methods of ProcessingInstruction

Table 16–12 Summary of Methods of ProcessingInstruction

Method	Description
getData()() on page 16-42	Returns the PI's data.
getTarget() on page 16-42	Returns the PI's target.
setData() on page 16-42	Sets the PI's data.

getData()

Description

Returns data for a processing instruction.

Syntax

```
DOMString getData();
```

getTarget()

Description

Returns a processing instruction's target value.

Syntax

```
DOMString getTarget();
```

setData()

Description

Sets the data for a processing instruction.

Syntax

```
void setData(DOMString data);
```

Parameter	Description
data	PI's new data

Text Class

Description of Text

This class contains methods for accessing and modifying the data associated with `Text` nodes, a subclasses of `CharacterData`.

Methods of Text

splitText()

Description

Gets data (value) of text node. Splits 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 becomes the next node after. Returns a pointer to the new text node.

Syntax

```
Text* splitText( unsigned long offset);
```

Parameter	Description
offset	split point

XMLParser Class

Description of XMLParser

This class contains top-level methods for invoking the parser and returning high-level information about a document.

Methods of XMLParser

Table 16–13 Summary of Methods of XMLParser

Methods	Description
context() on page 16-45	Retrieves the context.
createDocument() on page 16-45	Creates a Document node and returns a pointer to that node.
getContent() on page 16-46	Returns the content model for a node.
getDocType() on page 16-46	Returns a pointer to a “DocType” structure which describes the DTD
getDocument() on page 16-46	Returns a pointer to the root node of the document after a document has been successfully parsed.
getDocumentElement() on page 16-46	Returns a pointer to the root element (node) of the document after a document has been successfully parsed.
getEncoding() on page 16-47	Returns the name of the current document’s character encoding scheme, such as “ASCII”, “UTF8”, and so on.
isSingleChar() on page 16-47	Returns TRUE if the document is specified as standalone on the <?xml?> line, FALSE otherwise
isStandalone() on page 16-47	Sets the I/O callback functions for the given access method.
setAccess() on page 16-48	Validates the document.
validate() on page 16-49	Frees any memory used during the previous parse.
xmlclean() on page 16-50	Initialize XML parser.
xmlinit() on page 16-50	Initializes encoded XML parser.
xmlinitenc() on page 16-51	Parses a document.
xmlparse() on page 16-52	Parses a buffer.
xmlparseBuffer() on page 16-53	Parses a DTD.

Table 16–13 Summary of Methods of XMLParser (Cont.)

Methods	Description
xmlparseDTD() on page 16-54	Parses a document from a file.
xmlparseFile() on page 16-56	Parses a document from a file.
xmlparseStream() on page 16-57	Returns error location information.
xmlwhere() on page 16-58	Retrieves the context.
xmlterm() on page 16-59	Creates a Document node and returns a pointer to that node.

context()

Description

Retrieves the context.

Syntax

```
xmlctx* context();
```

createDocument()

Description

Creates a `Document` node and returns a pointer to that node. When DTD is not `NULL`, the `Node.ownerDocument` attribute is set to the document being created.

Syntax

```
Document* createDocument( DOMString uri,
                          DOMString qname,
                          DocumentType* dtd);
```

Parameter	Description
uri	namespace URI of the new document element
qname	qualified name of the new document element
dtd	document type (DTD)

getContent()

Description

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`	node whose content model to return

getDocType()

Description

Returns a pointer to a “DocType” structure which describes the DTD

Syntax

```
DocumentType* getDocType();
```

getDocument()

Description

Returns a pointer to the root node of the document after a document has been successfully parsed. Compare with `getDocumentElement` which returns the root *element* node.

Syntax

```
Node* getDocument();
```

getDocumentElement()

Description

Returns a pointer to the root element (node) of the document after a document has been successfully parsed.

Syntax

```
Element* getDocumentElement();
```

getEncoding()

Description

Returns the name of the current document's character encoding scheme, such as "`ASCII`", "`UTF8`", and so on. Compare to `isSingleChar` flag that only tells whether encoding is single or multibyte.

Syntax

```
DOMString getEncoding();
```

isSingleChar()

Description

Returns a flag which specifies whether the current document is encoded as single-byte characters, such as "`ASCII`", or multibyte characters, such as "`UTF-8`". Compare to `getEncoding` that returns the actual name of the document's encoding.

Syntax

```
boolean isSingleChar();
```

isStandalone()

Description

Returns `TRUE` if the document is specified as standalone on the `<?xml?>` line, `FALSE` otherwise

Syntax

```
boolean isStandalone();
```

setAccess()

Description

Sets the I/O callback functions for the given access method. Returns the error code, or `0` on success.

Most methods have built-in callback functions, so none are provided by the user. The notable exception is `XMLACCESS_STREAM`, user-defined streams, where the user *must* set the stream callback functions.

The three callback functions are invoked to open, close, and read from the input source. The functions should be declared using the function prototype macros `XML_OPENF`, `XML_CLOSEF` and `XML_READF`.

`XML_OPENF` is the open function, called once to open the input source. It should set its persistent handle in the `xmlihdl` union, which has two choices, a generic pointer `(void *)` or an integer (as unix file or socket handle). This function must return `XMLERR_OK` on success.

Parameter	IN / OUT	Description
ctx	(IN)	XML context
ih	(OUT)	the opened handle is placed here
length	(OUT)	total length of input source, if known; if not known, set to `0`
parts	(IN)	path broken down into components; opaque pointer
path	(IN)	full path to the source to be opened

`XML-CLOSEF` is the close function; it closes an open source and frees resources.

Parameter	IN / OUT	Description
ctx	(IN)	XML context
ih	(IN)	input handle union

`XML-READF` is the reader function; it reads data from an open source into a buffer, and returns the number of bytes read.

On EOI, the matching close function will be called automatically.

Parameter	IN / OUT	Description
ctx	(IN)	XML context
ih	(IN)	input handle union
dest	(OUT)	destination buffer into which data is read
destsize	(IN)	size of dest
path	(IN)	full path to the source to be opened; only provided here for use in error messages
nraw	(OUT)	number of bytes read
eoi	(OUT)	hit End of Information? Should be set to `TRUE` or `FALSE` after each read.

Syntax

```
uword setAccess( xmlctx *ctx,
                 xmlacctype access,
                 XML_OPENF((*openf)),
                 XML_CLOSEF( (*closef)),
                 XML_READF((*readf)));
```

Parameter	Description
ctx	the XML context
access	access method enum, `XMLACCESS_xxx`
openf	open-input callback function
closef	close-input callback function
readf	Read-input callback function

validate()

Description

Validates the document. Returns error code, `0` on success.

Syntax

```
uword validate( Node* root);
```

Parameter	Description
root	document node to validate

xmlclean()

Description

Frees any memory used during the previous parse. Recycles memory within the XML parser, but does not free it to the system -- only `xmlterm()` finally releases all memory back to the system. If `xmlclean()` is not called between parses, then the data used by the previous documents remains allocated, and pointers to it are valid. Thus, the data for multiple documents can be accessible simultaneously, although only the current document can be manipulated with DOM. To access only one document's data at a time, within a single context, call `xmlclean()` before each new parse.

Syntax

```
void xmlclean();
```

xmlinit()

Description

Initialize XML parser. Returns error code, `0` on success.

Syntax

```
uword xmlinit( DOMString incoding,
               void (*msghdlr)(void *msgctx, DOMString msg, ub4 errcode),
               void *msgctx,
               lpxsaxcb *saxcb,
               void *saxcbctx,
               DOMString lang);
```

Parameter	Description
incoding	Default input file encoding; UTF8 if not specified
msghdlr	Error message callback
msg	Message
errcode	Error code
msgctx	User-defined ocntext pointer passed to msghdr
saxcb	SAX callback structure (only if using SAX)
saxcbctx	User-defined SAX content structure passed to SAX callback functions
lang	Language of error message; currently not in use

xmlinitenc()

Description

Initializes encoded XML parser. Returns error code, 0 on success.

Syntax

```
uword xmlinitenc( DOMString incoding,
                  DOMString outcoding,
                  void (*msghdlr)(void *msgctx, DOMString msg, ub4 errcode),
                  void *msgctx,
                  lpxsaxcb *saxcb,
                  void *saxcbctx,
                  DOMString *lang);
```

Parameter	Description
incoding	Default input file encoding; UTF8 if not specified
outcoding	Output (DOM) encoding for document data; same as first input, if not specified
msghdlr	Error message callback
msgctx	User-defined pointer passed to msghdr
msg	Message
errcode	Error code

Parameter	Description
saxcb	SAX callback structure (only if using SAX)
saxcbctx	User-defined SAX content structure passed to SAX callback functions
lang	Language of error message; currently not in use

xmlparse()

Description

Parses a document. Returns error code, `0` on success.

Syntax

```
uword xmlparse( DOMString doc,
                DOMString encoding,
                ub4 flags);
```

Parameter	Description
doc	document path
encoding	document's encoding
flags	mask of flag bits

Parser Flag Option	Description
XML_DTD_ONLY	Parses an external subset. This is the same as calling `xmlparsedtd()`; it parses an external subset (DTD) instead of a complete XML document. Used primarily by the Class Generator so that it may generate classes from a DTD without need of a complete document.
XML_FLAG_DISCARD_WHITESPACE	Discards extraneous whitespace, such as end-of-line, and so on; default behavior is to report whitespace and indicate which whitespace can be ignored. This option will discard all whitespace between an end-element tag and the following start-element tag.

Parser Flag Option	Description
XML_FLAG_STOP_ON_WARNING	Stops validation on warnings. Validation problems are considered warnings (non-fatal) unless this flag is set. If set, validation will stop after the first warning.
XML_FLAG_VALIDATE	Turns validation on; default behavior is to only check for well-formedness.
XML_FORCE_INCODING	Forces input documents to be interpreted in the encoding “incoding”. The default input encoding may be specified as incoding, which overrides the incoding given to xmlinit. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be incoding. IANA / Mime encoding names should be used, “UTF-8”, “ASCII”, and so on. If `XML_FLAG_FORCE_INCODING` is set, the document will be interpreted as “incoding” regardless.
XML_WARN_DUPLICATE_ENTITY	Causes a warning to be emitted if a duplicate entity declaration is found. A duplicate entity declaration is usually silently ignored. When set, this flag causes a warning to be emitted instead.

xmlparseBuffer()

Description

Parses a buffer. Returns error code, `0` on success.

Syntax

```
uword xmlparseBuffer( DOMString buffer,
                      size_t len,
                      DOMString encoding,
                      ub4 flags);
```

Parameter	Description
buffer	buffer containing document to parse
len	length of document
encoding	document’s encoding
flags	mask of flag bits

Parser Flag Option	Description
XML_DTD_ONLY	Parses an external subset. This is the same as calling `xmlparsedtd()`; it parses an external subset (DTD) instead of a complete XML document. Used primarily by the Class Generator so that it may generate classes from a DTD without need of a complete document.
XML_FLAG_DISCARD_WHITESPACE	Discards extraneous whitespace, such as end-of-line, and so on; default behavior is to report whitespace and indicate which whitespace can be ignored. This option will discard all whitespace between an end-element tag and the following start-element tag.
XML_FLAG_STOP_ON_WARNING	Stops validation on warnings. Validation problems are considered warnings (non-fatal) unless this flag is set. If set, validation will stop after the first warning.
XML_FLAG_VALIDATE	Turns validation on; default behavior is to only check for well-formedness.
XML_FORCE_INCODING	Forces input documents to be interpreted in the encoding “incoding”. The default input encoding may be specified as incoding, which overrides the incoding given to xmlinit. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be incoding. IANA / Mime encoding names should be used, “UTF-8”, “ASCII”, and so on. If `XML_FLAG_FORCE_INCODING` is set, the document will be interpreted as “incoding” regardless.
XML_WARN_DUPLICATE_ENTITY	Causes a warning to be emitted if a duplicate entity declaration is found. A duplicate entity declaration is usually silently ignored. When set, this flag causes a warning to be emitted instead.

xmlparseDTD()

Description

Parses a DTD. Returns error code, `0` on success.

Syntax

```
uword xmlparseDTD( DOMString uri,
                   DOMString name,
                   DOMString encoding,
                    ub4 flags);
```

Parameter	Description
uri	URI pointing to DTD
name	DTD name
encoding	DTD’s encoding
flags	Mast of flag bits

Parser Flag Option	Description
XML_DTD_ONLY	Parses an external subset. This is the same as calling `xmlparsedtd()`; it parses an external subset (DTD) instead of a complete XML document. Used primarily by the Class Generator so that it may generate classes from a DTD without need of a complete document.
XML_FLAG_DISCARD_WHITESPACE	Discards extraneous whitespace, such as end-of-line, and so on; default behavior is to report whitespace and indicate which whitespace can be ignored. This option will discard all whitespace between an end-element tag and the following start-element tag.
XML_FLAG_STOP_ON_WARNING	Stops validation on warnings. Validation problems are considered warnings (non-fatal) unless this flag is set. If set, validation will stop after the first warning.
XML_FLAG_VALIDATE	Turns validation on; default behavior is to only check for well-formedness.
XML_FORCE_INCODING	Forces input documents to be interpreted in the encoding “incoding”. The default input encoding may be specified as incoding, which overrides the incoding given to xmlinit. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be incoding. IANA / Mime encoding names should be used, “UTF-8”, “ASCII”, and so on. If `XML_FLAG_FORCE_INCODING` is set, the document will be interpreted as “incoding” regardless.

Parser Flag Option	Description
XML_WARN_DUPLICATE_ENTITY	Causes a warning to be emitted if a duplicate entity declaration is found. A duplicate entity declaration is usually silently ignored. When set, this flag causes a warning to be emitted instead.

xmlparseFile()

Description

Parses a document from a file. Returns error code, `0` on success.

Syntax

```
uword xmlparseFile( DOMString path,
                    size_t len,
                    DOMString encoding,
                    ub4 flags);
```

Parameter	Description
path	document path
len	unused parameter
encoding	document’s encoding
flags	mask of flag bits

Parser Flag Option	Description
XML_DTD_ONLY	Parses an external subset. This is the same as calling `xmlparsedtd()`; it parses an external subset, DTD, instead of a complete XML document. Used primarily by the Class Generator so that it may generate classes from a DTD without need of a complete document.
XML_FLAG_DISCARD_WHITESPACE	Discards extraneous whitespace, such as end-of-line, and so on; default behavior is to report whitespace and indicate which whitespace can be ignored. This option will discard all whitespace between an end-element tag and the following start-element tag.

Parser Flag Option	Description
XML_FLAG_STOP_ON_WARNING	Stops validation on warnings. Validation problems are considered warnings (non-fatal) unless this flag is set. If set, validation will stop after the first warning.
XML_FLAG_VALIDATE	Turns validation on; default behavior is to only check for well-formedness.
XML_FORCE_INCODING	Forces input documents to be interpreted in the encoding "incoding". The default input encoding may be specified as incoding, which overrides the incoding given to xmlinit. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be incoding. IANA / Mime encoding names should be used, "UTF-8", "ASCII", and so on. If `XML_FLAG_FORCE_INCODING` is set, the document will be interpreted as "incoding" regardless.
XML_WARN_DUPLICATE_ENTITY	Causes a warning to be emitted if a duplicate entity declaration is found. A duplicate entity declaration is usually silently ignored. When set, this flag causes a warning to be emitted instead.

xmlparseStream()

Description

Parses a document from a file. Returns error code, `0` on success.

Syntax

```
uword xmlparseStream( DOMString path,
                      void *stream,
                      DOMString encoding,
                      ub4 flags);
```

Parameter	Description
path	unused parameter
stream	input stream
encoding	document's encoding
flags	mask of flag bits

Parser Flag Option	Description
XML_DTD_ONLY	Parses an external subset. This is the same as calling `xmlparsedtd()`; it parses an external subset, DTD, instead of a complete XML document. Used primarily by the Class Generator so that it may generate classes from a DTD without need of a complete document.
XML_FLAG_DISCARD_WHITESPACE	Discards extraneous whitespace, such as end-of-line, and son on; default behavior is to report whitespace and indicate which whitespace can be ignored. This option will discard all whitespace between an end-element tag and the following start-element tag.
XML_FLAG_STOP_ON_WARNING	Stops validation on warnings. Validation problems are considered warnings (non-fatal) unless this flag is set. If set, validation will stop after the first warning.
XML_FLAG_VALIDATE	Turns validation on; default behavior is to only check for well-formedness.
XML_FORCE_INCODING	Forces input documents to be interpreted in the encoding "incoding". The default input encoding may be specified as incoding, which overrides the incoding given to xmlinit. If the input's encoding cannot be determined automatically, based on BOM, XMLDecl, and others, then it is assumed to be incoding. IANA / Mime encoding names should be used, "UTF-8", "ASCII", and so on. If `XML_FLAG_FORCE_INCODING` is set, the document will be interpreted as "incoding" regardless.
XML_WARN_DUPLICATE_ENTITY	Causes a warning to be emitted if a duplicate entity declaration is found. A duplicate entity declaration is usually silently ignored. When set, this flag causes a warning to be emitted instead.

xmlwhere()

Description

Returns error location information. Should only be called from within user error calback function, while error is current.

Syntax

```
boolean xmlwhere( ub4 *line,
                  DOMString *path,
                    uword idx);
```

Parameter	Description
line	returned line # where error occurred
path	returned path / URL where error occurred
idx	position in error stack, starting at 0

xmlterm()

Description

Terminates XML parser; tear down and free memory.

Syntax

```
void xmlterm();
```

C++ SAX APIs

Description of 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 nonzero value meaning failure. If a nonzero 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.

Note this SAX functionality is identical to the C version.

SAX Callback Structure

```
typedef struct  {
sword (*startDocument) (void *ctx);
sword (*endoocument) (void *ctx);
sword (*startElement) (void *ctx, const oratext *name,
        const struct xmlnodes *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 *public const oratext *systemId,
        const oratext *notationName);
sword (*nsStartElement) (void *ctx, const oratext *qname,
        const oratext *local, const oratext *nsp,
        const struct xmlnodes *attrs);
sword (*comment) (void *ctx, const oratext *data);
sword (*elementDecl) (void *ctx, const oratext *name,
        const oratext *content);
sword (*attributeDecl) (void *ctx, const oratext *elem,
        const oratext *attr, const oratext *body);
} xmlsaxcb;
```

Methods of C++ SAX API

Table 16–14 Summary of Methods of C++ SAX API

Method	Description
startDocument() on page 16-61	Starts document processing; called only once.
endDocument() on page 16-62	Finishes document processing; called only once.
startElement() on page 16-62	Starts processing a new document element; called only once for each element.
endElement() on page 16-63	Finishes processing a document element; called only once for each element.
characters() on page 63	Processes literal text; called for each piece of literal text.
ignorableWhitespace() on page 16-64	Processes ignorable (non-significant) whitespace; called for each piece of ignorable whitespace.
processingInstruction() on page 16-64	Processes PIs (Processing Instructions); called once for each PI.
notationDecl on page 16-65	Processes notation; called once for each NOTATION.
unparsedEntityDecl() on page 16-65	Processes unparsed entity declarations; called once for each unparsed entity declaration.
nsStartElement() on page 16-66	Starts processing a new document element when the element uses an explicit namespace; called only once for each element.
comment() on page 16-66	Receives notification about an XML source comment.
elementDecl() on page 16-67	Receives notification about an element declaration.
attributeDecl() on page 16-67	Receives notification about an element's attribute declaration.

startDocument()

Description

Starts document processing; called only once. Returns `0` for success or a numeric error code.

Syntax

```
sword startDocument( void *ctx);
```

Parameter	Description
ctx	User-defined context as passed to `initialize()`.

endDocument()

Description

Finishes document processing; called only once. Returns `0` for success or a numeric error code.

Syntax

```
sword endDocument( void *ctx);
```

Parameter	Description
ctx	User-defined context as passed to `initialize()`.

startElement()

Description

Starts processing a new document element; called only once for each element. Returns `0` for success or a numeric error code.

Syntax

```
sword startElement(  void *ctx,
                     const oratext *name,
                     const struct xmlnodes *attrs);
```

Parameter	Description
ctx	User-defined context as passed to `initialize()`.
name	name of node
attrs	array of node's attributes

endElement()

Description

Finishes processing a document element; called only once for each element. Returns `0` for success or a numeric error code.

Syntax

```
sword endElement( void *ctx,
                  const oratext *name);
```

Parameter	Description
ctx	User-defined context as passed to `initialize()`.
name	name of node

characters()

Description

Processes literal text; called for each piece of literal text. Returns `0` for success or a numeric error code.

Syntax

```
sword characters( void *ctx,
                  const oratext *ch,
                  size_t len);
```

Parameter	Description
ctx	User-defined context as passed to `initialize()`
ch	pointer to text
len	number of characters in text

ignorableWhitespace()

Description

Processes ignorable (non-significant) whitespace; called for each piece of ignorable whitespace. Returns `0` for success or a numeric error code.

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 in whitespace text

processingInstruction()

Description

Processes PIs (Processing Instructions); called once for each PI. Returns `0` for success or a numeric error code.

Syntax

```
sword processingInstruction( void *ctx,
                             const oratext *target,
                             const oratext *data);
```

Parameter	Description
ctx	User-defined context as passed to `initialize()`
data	PI data
target	PI target

notationDecl

Description

Processes notation; called once for each `NOTATION`. Returns `0` for success or a numeric error code.

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()

Description

Processes unparsed entity declarations; called once for each unparsed entity declaration. Returns `0` for success or a numeric error code.

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()`

Parameter	Description
name	name of entity
publicId	Public ID
systemID	System ID
notationName	notation name

nsStartElement()

Description

Starts processing a new document element when the element uses an explicit namespace; called only once for each element. Returns `0` for success or a numeric error code.

Syntax

```
sword startElement( void *ctx,
                    const oratext *qname,
                    const oratext *local,
                    const oratext *namespace,
                    const struct xmlnodes *attrs);
```

Parameter	Description
ctx	User-defined context as passed to `initialize()`
qname	qualified namespac
local	element local name
namespace	element namespace
attrs	specified or default attributes

comment()

Description

Receives notification about an XML source comment. Returns `0` for success or a numeric error code.

Syntax

```
sword comment( void *ctx,
               const oratext *data);
```

Parameter	Description
ctx	User-defined context as passed to `initialize()`
ldata	body of comment

elementDecl()

Description

Receives notification about an element declaration. Returns `0` for success or a numeric error code.

Syntax

```
sword elementDecl( void *ctx,
                   const oratext *name,
                   const oratext *content);
```

Parameter	Description
ctx	User-defined context as passed to `initialize()`
name	name of element being declared
content	content model for element

attributeDecl()

Description

Receives notification about an element's attribute declaration. Returns `0` for success or a numeric error code.

Syntax

```
sword attributeDecl( void *ctx,
                          const oratext *elem,
                          const oratext *attr,
                          const oratext *body);
```

Parameter	Description
ctx	User-defined context as passed to `initialize()`
elem	name of element for which the attribute is declared
attr	name of attribute declared
body	body of attribute declaration

C++ DOM API's

Table 16–15 Classes of C++ DOM API

Class	**Superclass**
Attribute	Node
CDATASection	Text
CharacterData	Node
Comment	CharacterData
Document	Node
DocumentFragment	Node
DocumentType	Node
Element	Node
Entity	Node
EntityReference	Node
Notation	Node
ProcessingInstruction	Node
Text	CharacterData

17

XSLT Processor for C++

This chapter contains the following sections:

- XSLProcessor Class
- XPObject Class
- XPath Class

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*

XSLProcessor Class

Description of XSLProcessor

This class contains top-level methods for invoking the XSL processor.

Methods of XSL Processor

xslprocess()

Description

Processes an XSL stylesheet with an XML document source. Returns a numeric error code, or 0 on success.

Syntax

```
uword xslprocess( XMLParser *docctx,
                  XMLParser *xslctx,
                  XMLParser *resctx,
                  Node **result);
```

Argument	IN / OUT	Description
docctx	(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

XPObject Class

Description of XPObject

This class contains top-level methods of class XPObject. Objects of this class are created as a result of XPath expression evaluation.

Methods of XPObject

Table 17–1 Summary of Methods of XPObject

Method	Description
XPObject() on page 17-3	XPObject constructor.
getbooleanval() on page 17-3	Returns boolean value of XPObject.
getnumval() on page 17-4	Returns node set value of XPObject.
getstrval() on page 17-4	Returns numeric value of XPObject.
getnsetval() on page 17-4	Returns string value of XPObject.
getxpobjtyp() on page 17-4	Returns type of XPObject.

XPObject()

Description

XPObject constructor.

Syntax

```
XPObject();
```

getbooleanval()

Description

Returns boolean value of the object.

Syntax

```
boolean getbooleanva();
```

getnumval()

Description

Returns numeric value of the object.

Syntax

```
double getnumval();
```

getstrval()

Description

Returns string value of the object.

Syntax

```
oratext* getstrval();
```

getnsetval()

Description

Returns node set value of the object.

Syntax

```
xpnset* getnsetval();
```

getxpobjtyp()

Description

Returns the object type. The returned type could be one of the following:

- XPOBJTYP_BOOL
- XPOBJTYP_NUM
- XPOBJTYP_STR
- XPOBJTYP_NSET
- XPOBJTYP_RTFRAG

Syntax

```
xpobjtyp getxpobjtyp();
```

XPath Class

Description of XPath

This class contains top-level methods for invoking the XPath processor.

Methods of XPath

Table 17–2 Summary of Methods of XPath

Method	Description
XPath() on page 17-6	XPath constructor.
~XPath on page 17-7	XPath destructor.
parsexpathexpr() on page 17-7	Parses an XPath expression.
evalxpathexpr() on page 17-7	Evaluates a previously parsed XPath expression.

XPath()

Description

Creates and returns an XPath object. This call never fails.

Syntax

```
XPath( xmlctx *ctx,
       xmlnode *xslnode,
       xmlnode* xml_node,
       oratext* baseURI,
       size_t nctxels,
       xmlnode** ctxnodes);
```

Argument	IN / OUT	Description
ctx	(IN)	XSL context. Could be `NULL`.
xslnode	(IN)	The XSL node to be used for namespace expansion. Could be set to `NULL`.
xml_node	(IN)	The context node. Set to `NULL` for parsing.
baseURI	IN)	The base URI for parsing.

Argument	IN / OUT	Description
nctxels	(IN)	The number of nodes in the current node set.
ctxnodes	(IN)	The current node set.

~XPath

Description

Deletes / destroys an XPath object.

Syntax

```
delete();
```

parsexpathexpr()

Description

Parses an XPath expression. Returns the expression tree on success, otherwise `NULL`.

Syntax

```
xpexpr *parsexpathexpr( oratext *expr,
                        sword *err);
```

Argument	IN / OUT	Description
expr	(IN)	The expression in the form of a string
err	(OUT)	The error code

evalxpathexpr()

Description

Evaluates an XPath expression. Returns the result object on success, or `NULL` on failure. The type of the result object, obtained by calling `getxpobjtyp()`, is one of:

- `XPOBJTYP_BOOL`
- `XPOBJTYP_NUM`
- `XPOBJTYP_STR`

- `XPOBJTYP_NSET`
- `XPOBJTYP_RTFRAG`

Syntax

```
XPObject *evalxpathexpr( xpexpr *exprtree,
                         sword *err);
```

Argument	IN / OUT	Description
expr	(IN)	The expression in the form of a tree
err	(OUT)	The error code

18

XML Schema Processor for C++

This chapter contains the following sections:

- Description of XMLSchema
- Functions of XMLSchema

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*

XMLSchema Class

Description of XMLSchema

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 nonzero 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 the XML parser function `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.

Functions of XMLSchema

Table 18–1 Summary of Functions of XMLSchema

Function	Description
initialize() on page 18-2	Initializes Schema processor.
load() on page 18-3	Loads an additional Schema file.
target() on page 18-3	Returns target namespace URI.
terminate() on page 18-4	Terminates Schema Processor.
validate() on page 18-4	Validates an instance document against a Schema.

initialize()

Description

Initializes Schema processor. Returns error code, or `0` for success.

Syntax

```
uword initialize( XMLParser *parser);
```

Argument	Description
parser	XMLParser object used for allocating memory.

load()

Description

Explicitly loads a schema (if not already loaded); returns a pointer to the schema, new or old. It is not usually necessary to use this method, as the URL of a default schema may be passed directly to the validate method.   Returns error code, `0` for success.

Syntax

```
uword load( oratext *uri,
            oratext *nsp,
            Schema **schema);
```

Argument	Description
uri	URI of Schema, in compiler character set
nsp	Namespace of Schema (may be `NULL`), in compiler character set.
schema	Returned pointer to schema.

target()

Description

Return its target namespace URI given a Schema handle that is returned by `load()`.

Syntax

```
oratext *target( Schema *schema);
```

Argument	Description
schema	Schema handle as returned by `load()` method.

terminate()

Description

Terminate Schema processor, tear down, free memory, and so on. Returns error code, `0` for success

Syntax

```
void terminate();
```

validate()

Description

Validates an instance document against a Schema.

Syntax

```
uword validate( Element *root,
                oratext *url);
```

Argument	Description
root	root element of the document returned by `parser.getDocuemntElement()`
url	URL of default schema; optional

19

XML Class Generator for C++

This chapter contains the following sections:

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*

Overview of the XML Class Generator for C++

The XML Class Generator takes a Document Type Definition (DTD) or XML Schema and generates classes for each defined element. Those classes are then used in a C++ program to construct XML documents conforming to the DTD.

Input

Input is an XML document containing a DTD, an external DTD, or an XML Schema. IF a complete XML document is provided, the document body itself is ignored; only the DTD is relevant, though the dummy document must conform to the DTD.

Output

Output is a pair of C++ source files, .cpp and .h, named after the DTD if a complete XML document is provided. For an external DTD or Schema, the name of the generated files must be provided. 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 XML Standards

The W3C recommendation for Extensible Markup Language (XML) 1.0

The W3C recommendation for Document Object Model Level 1 1.0

The W3C proposed recommendation for Namespaces in XML

The Simple API for XML (SAX) 1.0

Sample Usage

The standalone parser may be called as an executable by invoking

```
bin/xmlcg like
xmlcg [flags] <XML document>
```

Table 19–1 Optional Flags

Flag	Command	Description
-d	directory	Specify output directory; default is current directory
-e	encoding	Specify default input file encoding
-h	help	show this usage help

XMLClassGenerator Class

Description of XMLClassGenerator

This class contains the methods for generating classes based on a DTD or Schema.

Methods of XMLClassGenerator

generate()

Description

Generates classes for a given DTD or Schema. Returns error code, or `0` on success. The options are described in the following table.

Syntax	Description
uword XMLClassGenerator::generate(DocumentType *dtd, DOMString outdir);	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.
uword XMLClassGenerator::generate(Schema*schema,DOMStringname, DOMString outdir);	Generates classes for the given Schema. Two files are created in the output directory `outdir` (or in the correct directory if `outdir` is `NULL`): `name.h` and `name.cpp`. One class is generated for each element in the Schema.

Parameter	Description
dtd	Classes will be generated for elements of this DTD.
outdir	Directory where output files will be placed.
schema	Classes will be generated for elements of this Schema.
name	Name for generated files and enclosing C++ namespace.

generated Class (for DTD)

Description of generated

A generated class is produced for each element defined in the DTD, with the same name as the element.

If an element (or attribute) name cannot be used directly as a C++ identifier, it is mapped to a valid identifier by converting it to the compiler character set (ASCII or EBCDIC) and then replacing unmappable characters with the two-letter hex for their code points. For example, the element name "Curaçao" maps to "Curacao". If the remapped name is already used, digits are appended to the end to make it unique; for example, "Curacao0", and so on. Note that elements and attributes created by the generated classes will have the original names. The remapping only applies to the generated code itself, so that it will be syntactically correct in C++. It does not apply to the XML elements and data that are constructed by the these elements.

There are two styles of creation: making an empty element and then adding the children one at a time, or constructing the element with initial data or children. For example, given the element declaration:

```
<!ELEMENT B (#PCDATA | F)*>
```

The following constructors will be provided:

Constructor	Description
B(Document *doc);	makes an empty element with no children
B(Document *doc, String data);	initializes it with PCDATA
(Document *doc, F *theF);	initialized with a single child node of element F

An element like B, that may contain `PCDATA`, can also add the data post-construction using:

```
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=newF(...);
b=new B(f);
```

and

```
f=newF(...);
b=newE();
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 the desired final elements cannot be made using one of the forms that take initial children, an empty element must be declared first so nodes can be added as needed with `addNode()`.

For each attribute for an element, a method is provided to set its value, named `set`*`attrname`*`()`. For example, for the element declaration,

```
<!ELEMENT D (#PCDATA)>
<!ATTLIST D foo CDATA #REQUIRED>
```

class D will have the method

```
 Attr* setfoo(String value);
```

Note: The constructed element is not tested for validity as it is being made. The user must explicitly call the XMLParser's `validate()` method on the final element.

Methods of generated

Table 19–2 Summary of Methods of generated (for DTD)

Method	Description
class() on page 19-7	Constructor.
addData() on page 19-7	Adds PCDATA to the element.
addNode() on page 19-8	Adds a node to the element.
setattribute() on page 19-8	Sets one of the element's attributes.

class()

Description

Class constructor. Constructs an element which will belong to the given document. See the example given at the beginning of this section. The options are described in the following table.

Syntax	Description
class(Document *doc);	Makes the element with no children; requires use `addData()` and `addNode()` as appropriate to fill it out.
class(Document *doc, ...);	Used to provide initial data or children, the exact choices of which depend on the element definition.

Parameter	Description
doc	Document to which the element belongs.
...	Varying arguments, depending on the element definition.

addData()

Description

Adds data to the element by appending 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 using `XMLParser::normalize()`.

Syntax

```
void addData( Document *doc,
              String data);
```

Parameter	Description
doc	Document to which the element belongs.
data	Data to be added.

addNode()

Description

Adds/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()`.

Prototype

```
void addNode( node thenode);
```

Parameter	Description
thenode	Node to be added.

set*attribute*()

Description

Sets the element's attribute with the given value. One method is provided for each attribute, named after the attribute as `set`*`attribute`*`()`. Returns the created attribute.

Prototype

```
Attr* setattribute( String value);
```

Parameter	Description
value	The attribute's value.

generated Class (for Schema)

Description of generated

All generated classes are enclosed in a C++ namespace with the same name as the generated files; for example, `Foo.cpp` will contain namespace `Foo`.

A *generated* class is produced for each element in the Schema. It has the same name as the element, except for local elements which have the parent element's name prefixed.

If an element (or attribute) name cannot be used directly as a C++ identifier, it is mapped to a valid identifier by converting it to the compiler character set (ASCII or EBCDIC) and then replacing unmappable characters with the two-letter hex for their code points. For example, the element name “Curaçao” maps to “Curacao”. If the remapped name is already used, digits are appended to the end to make it unique; for example, “Curacao0”, and so on. Note that elements and attributes created by the generated classes will have the original names. The remapping only applies to the generated code itself, so that it will be syntactically correct in C++, not to the XML elements and data which are constructed by them.

There are two styles of creation: making an empty element and then adding the children one at a time, or constructing the element with initial data or children. For example, the following constructors will be provided given the element declaration:

```
<element name="foo">
   <complexType content="mixed">
      <element ref="thing" minOccurs="0"/>
      <attribute name="bar" use="required" type="int"/>
   </complexType>
</element>
```

the following constructors will be provided:

```
foo(Document *doc);
     // Makes an empty element with no children

foo(Document *doc, DOMString s);
     // Initializes it with PCDATA

foo(Document *doc, Que::thing *the_thing);
     // Initialized with a single child node of element thing
```

An element like `foo` that may contain PCDATA is also given a method to add the data post-construction:

```
void foo::addData(Document *doc, DQMString s);
```

Each possible child element of `foo` also is given an “assembler”:

```
void foo::addNode(Queue::thing *the_thing);
```

The following usages are equivalent:

```
f = new Queue::foo("data");
```

and

```
f = new Queue::foo();
f->addData("data");
```

Similarly, the following are also equivalent:

```
f = new Queue::foo(...);
t = new Queue::thing(...);
```

and

```
f = new Queue::foo(...);
t = new Queue::thing(...);
f->addNode(t);
```

Not all possible combinations of initial elements are provided constructors, especially considering variable occurrences. If no constructor is appropriate, the element must be built-up. For example, consider the element definition

```
<element name="map-data">
   <complexType content="mixed">
      <element ref="aq:item" minOccurs="0" maxOccurs="*"/>
   </complexType>
</element>
```

A `map-data` element may contain any number of `aq:item` children. In such cases, a constructor is provided which allows only one occurrence; additional occurrences must be assembled, like the following example which needs four:

```
md = new Queue::map~data(doc, i1);
md->addNode(i2);
md->addNode(i3);
md->addNode(i4);
```

For each attribute of an element, a method is provided to set its value, named `set_attrname()`. For example, for the element declaration,

```
<element name="client-operation">
   <complexType content="mixed">
      <element name="txid" type="string" minOccurs="0"/>
      <attribute name="opcode" use="required" type="aq:opcode_type"/>
   </complexType>
</element>
```

a method would be provided to set the attribute:

```
Attr* client_operation::set_opcode(DOMString s);
```

Note: The constructed element is not tested for validity as it is being made. The user must explicitly call the XMLSchema's validate method on the final element.

Methods of generated

Table 19–3 Summary of Methods of generated (for Schema)

Method	Description
class() on page 19-11	Class constructor.
addData() on page 19-12	Adds PCDATA to the element
addNode() on page 19-12	Adds a node to the element
set_attribute() on page 19-13	Sets one of the element’s attributes

class()

Description

Constructs an element which will belong to the given document. See the example given at the beginning of this section. The options are described in the following table.

Syntax	Description
class(Document *doc);	Makes the element with no children; requires use `addData()` and `addNode()` as appropriate to fill it out.

Syntax	Description
class(Document *doc, ...);	Used to provide initial data or children, the exact choices of which depend on the element definition.

Parameter	Description
doc	Document to which the element belongs.
...	Varying arguments, depending on the element definition.

addData()

Description

Adds data to the element by appending 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 using `XMLParser::normalize()`.

Syntax

```
void addData( Document *doc,
              String data);
```

Parameter	Description
doc	Document to which the element belongs.
data	Data to be added

addNode()

Description

Adds/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 the_node);
```

Parameter	Description
the_node	The node to be added.

set_*attribute*()

Description

Sets the element's attribute with the given value. Returns the created attribute. One method is provided for each attribute, named after the attribute as `set_`*`attribute`*`()`.

Syntax

```
Attr* set_attribute( String value);
```

Part IV

XDK for PL/SQL

This section contains the following chapters:

20 XML SQL Utility (XSU) for PL/SQL

XML SQL Utility (XSU) for PL/SQL consists of two packages:

- DBMS_XMLQuery Package
- DBMS_XMLSave Package

See Also:

- *Oracle9i XML Developer's Kits Guide - XDK*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

DBMS_XMLQuery Package

Description of DBMS_XMLQuery

This API provides DB_to_XML type functionality.

Types of DBMS_XMLQuery

Table 20–1 Types of DBMS_XMLQuery

Type	Description
ctxType	The type of the query context handle. This is the return type of newContext().

Constants of DBMS_XMLQuery

Table 20–2 Constants of DBMS_XMLQuery

Constant	Description
DB_ENCODING	Used to signal that the DB character encoding is to be used.
DEFAULT_ROWSETTAG	The tag name for the element enclosing the XML generated from the result set (that is, for most cases the root node tag name) -- ROWSET.
DEFAULT_ERRORTAG	The default tag to enclose raised errors -- ERROR.
DEFAULT_ROWIDATTR	The default name for the cardinality attribute of XML elements corresponding to db. records. -- NUM
DEFAULT_ROWTAG	The default tag name for the element corresponding to db. records. -- ROW
DEFAULT_DATE_FORMAT	Default date mask. -- 'MM/dd/yyyy HH:mm:ss'
ALL_ROWS	The ALL_ROWS parameter is to indicate that all rows are needed in the output.
NONE	Used to specifies that the output should not contain any XML metadata (for example, no DTD or Schema).
DTD	Used to specify that the generation of the DTD is desired.
SCHEMA	Used to specify that the generation of the XML SCHEMA is desired.
LOWER_CASE	Use lower cased tag names.

Table 20–2 Constants of DBMS_XMLQuery

Constant	Description
UPPER_CASE	Use upper case tag names.

Functions and Procedures of DBMS_XMLQuery

Table 20–3 Summary of Functions and Procedures of DBMS_XMLQuery

Functions/Procedures	Description
newContext() on page 20-4	Creates a query context and it returns the context handle.
closeContext() on page 20-5	Closes / deallocates a particular query context.
setRowsetTag() on page 20-5	Sets the tag to be used to enclose the XML dataset.
setRowTag() on page 20-6	Sets the tag to be used to enclose the XML element corresponding to a db.
setErrorTag() on page 20-6	Sets the tag to be used to enclose the XML error docs.
setRowIdAttrName() on page 20-6	Sets the name of the id attribute of the row enclosing tag.
setRowIdAttrValue() on page 20-7	Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag.
setCollIdAttrName() on page 20-7	Sets the name of the id attribute of the collection element's separator tag.
useNullAttributeIndicator() on page 20-8	Specifies weather to use an XML attribute to indicate `NULL`ness.
useTypeForCollElemTag() on page 20-8	Tells the XSU to use the collection element’s type name as the collection element tag name.
setTagCase() on page 20-9	Specified the case of the generated XML tags.
setDateFormat() on page 20-9	Sets the format of the generated dates in the XML doc.
setMaxRows() on page 20-10	Sets the max number of rows to be converted to XML.
setSkipRows() on page 20-10	Sets the number of rows to skip.
setStylesheetHeader() on page 20-10	Sets the stylesheet header.
setXSLT() on page 20-11	Registers a stylesheet to be applied to generated XML.
setXSLTParam() on page 20-12	Sets the value of a top-level stylesheet parameter.
removeXSLTParam() on page 20-12	Removes a particular top-level stylesheet parameter.

Table 20–3 Summary of Functions and Procedures of DBMS_XMLQuery (Cont.)

Functions/Procedures	Description
setBindValue() on page 20-13	Sets a value for a particular bind name.
setMetaHeader() on page 20-13	Sets the XML meta header.
setDataHeader() on page 20-14	Sets the XML data header.
setEncodingTag() on page 20-14	Sets the encoding processing instruction in the XML document.
setRaiseException() on page 20-15	Tells the XSU to throw the raised exceptions.
setRaiseNoRowsException() on page 20-15	Tells the XSU to throw or not to throw an `OracleXMLNoRowsException` in the case when for one reason or another, the XML doc generated is empty.
setSQLToXMLNameEscaping() on page 20-16	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.
propagateOriginalException() on page 20-16	Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an OracleXMLSQLException.
getExceptionContent() on page 20-17	Returns the thrown exception's error code and error message.
getDTD() on page 20-17	Generates the DTD.
getNumRowsProcessed() on page 20-18	Returns the number of rows processed for the query.
getVersion() on page 20-18	Prints the version of the XSU in use.
getXML() on page 20-18	Generates the XML document.

newContext()

Description

Creates a query context and it returns the context handle. The options are described in the following table.

Syntax	Description
FUNCTION newContext(sqlQuery IN VARCHAR2) RETURN ctxType	Creates a query context from a string.

Syntax	Description
FUNCTION newContext(sqlQuery IN CLOB) RETURN ctxType	Creates a query context from a CLOB.

Parameter	IN / OUT	Description
sqlQuery	(IN)	SQL query, the results of which to convert to XML.

closeContext()

Description

Closes / deallocates a particular query context

Syntax

```
PROCEDURE closeContext( ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

setRowsetTag()

Description

Sets the tag to be used to enclose the XML dataset.

Syntax

```
PROCEDURE setRowsetTag(ctxHdl IN ctxType, tag IN VARCHAR2)
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

setRowTag()

Description

Sets the tag to be used to enclose the XML element corresponding to a db. record.

Syntax

```
PROCEDURE setRowTag(ctxHdl IN ctxType, tag IN VARCHAR2)
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

setErrorTag()

Description

Sets the tag to be used to enclose the XML error docs.

Syntax

```
PROCEDURE setErrorTag( ctxHdl IN ctxType,
                       tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

setRowIdAttrName()

Description

Sets the name of the id attribute of the row enclosing tag. Passing null or an empty string for the tag results the row id attribute to be omitted.

Syntax

```
PROCEDURE setRowIdAttrName( ctxHdl IN ctxType,
                              attrName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

setRowIdAttrValue()

Description

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 for the colName results the row id attribute being assigned the row count value (that is, 0, 1, 2 and so on).

Syntax

```
PROCEDURE setRowIdAttrValue( ctxHdl IN ctxType,
                             colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column whose value is to be assigned to the row id attribute.

setCollIdAttrName()

Description

Sets the name of the id attribute of the collection element's separator tag.

Syntax

```
PROCEDURE setCollIdAttrName( ctxHdl IN ctxType,
                             attrName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
attrName	(IN)	AttributeName.

useNullAttributeIndicator()

Description

Specified weather to use an XML attribute to indicate NULLness, or to do it by omitting the inclusion of the particular entity in the XML document.

Syntax

```
PROCEDURE useNullAttributeIndicator( ctxHdl IN ctxType,
                                     flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Use attribute to indicate NULL?

useTypeForCollElemTag()

Description

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 of TRUE, tells the XSU to use the collection element's type name as the collection element tag name.

Syntax

```
PROCEDURE useTypeForCollElemTag( ctxHdl IN ctxType,
                                 flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Turn on use of the type name?.

setTagCase()

Description

Specified the case of the generated XML tags.

Syntax

```
PROCEDURE setTagCase( ctxHdl IN ctxType,
                      tCase IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tCase	(IN)	The tag’s case; 0=asAre, 1=lower, 2=upper.

setDateFormat()

Description

Sets the format of the generated dates in the XML doc. The syntax of the date format pattern (that is, the date mask), should conform to the requirements of the java.text.SimpleDateFormat class. Setting the mask to null or an empty string, results the use of the default mask -- DEFAULT_DATE_FORMAT.

Syntax

```
PROCEDURE setDateFormat( ctxHdl IN ctxType,
                         mask IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
mask	(IN)	The date mask.

setMaxRows()

Description

Sets the max number of rows to be converted to XML. By default there is no max set.

Syntax

```
PROCEDURE setMaxRows ( ctxHdl IN ctxType,
                       rows IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
rows	(IN)	Maximum number of rows to generate.

setSkipRows()

Description

Sets the number of rows to skip. By default 0 rows are skipped.

Syntax

```
PROCEDURE setSkipRows( ctxHdl IN ctxType,
                       rows IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
rows	(IN)	Maximum number of rows to skip.

setStylesheetHeader()

Description

Sets the stylesheet header (that is, stylesheet processing instructions) in the generated XML doc. Note: Passing null for the uri argument will unset the stylesheet header and the stylesheet type.

Syntax

```
PROCEDURE setStylesheetHeader( ctxHdl IN ctxType,
                               uri IN VARCHAR2,
                               type IN VARCHAR2 := 'text/xsl');
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
uri	(IN)	Stylesheet URI.
type	(IN)	Stylesheet type; defaults to "`text/xsl`".

setXSLT()

Description

Registers a stylesheet to be applied to generated XML. If a stylesheet was already registered, it gets replaced by the new one. The options are described in the following table.

Syntax	Description
PROCEDURE setXSLT(ctxHdl IN ctxType, uri IN VARCHAR2, ref IN VARCHAR2 := null);	To un-register the stylesheet pass in a null for the uri.
PROCEDURE setXSLT(ctxHdl IN ctxType, stylesheet CLOB, ref IN VARCHAR2 := null);	To un-register the stylesheet pass in a null or an empty string for the stylesheet.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
uri	(IN)	Stylesheet URI.
stylesheet	(IN)	Stylesheet.
ref	(IN)	URL to include, import and external entities.

setXSLTParam()

Description

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). NOTE: if no stylesheet is registered, this method is a no op.

Syntax

```
PROCEDURE setXSLTParam( ctxHdl IN ctxType,
                        name IN VARCHAR2,
                        value IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Name of the top level stylesheet parameter.
value	(IN)	Value to be assigned to the stylesheet parameter.

removeXSLTParam()

Description

Removes the value of a top-level stylesheet parameter. NOTE: if no stylesheet is registered, this method is a no op.

Syntax

```
PROCEDURE removeXSLTParam( ctxHdl IN ctxType,
                           name IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Name of the top level stylesheet parameter.

setBindValue()

Description

Sets a value for a particular bind name.

Syntax

```
PROCEDURE setBindValue( ctxHdl IN ctxType,
                        bindName IN VARCHAR2,
                        bindValue IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
bindName	(IN)	Bind name.
bindValue	(IN)	Bind value.

setMetaHeader()

Description

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. Note that the last meta header specified is the one that is used; furthermore, passing in null for the header, parameter unsets the meta header.

Syntax

```
PROCEDURE setMetaHeader( ctxHdl IN ctxType,
                         header IN CLOB := null);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
Header	(IN)	Header.

setDataHeader()

Description

Sets the XML data header. The data header is an 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. Note that the last data header specified is the one that is used; furthermore, passing in null for the header, parameter unsets the data header.

Syntax

```
PROCEDURE setDataHeader( ctxHdl IN ctxType,
                   header IN CLOB := null,
                   tag IN VARCHAR2 := null);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
header	(IN)	Header.
tag	(IN)	Tag used to enclose the data header and the rowset.

setEncodingTag()

Description

Sets the encoding processing instruction in the XML document.

Syntax

```
PROCEDURE setEncodingTag( ctxHdl IN ctxType,
                          enc IN VARCHAR2 := DB_ENCODING);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
enc	(IN)	The encoding to use.

setRaiseException()

Description

Tells the XSU 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 out of the exception's message.

Syntax

```
PROCEDURE setRaiseException( ctxHdl IN ctxType,
                             flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Throw raised exceptions? `TRUE` for yes, otherwise `FALSE`.

setRaiseNoRowsException()

Description

Tells the XSU whether to throw an OracleXMLNoRowsException in the case when for one reason or another, the XML doc generated is empty. By default, the exception is not thrown.

Syntax

```
PROCEDURE setRaiseNoRowsException( ctxHdl IN ctxType,
                                   flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Throw OracleXMLNoRowsException if no data? `TRUE` for yes, otherwise `FALSE`.

setSQLToXMLNameEscaping()

Description

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

```
PROCEDURE setSQLToXMLNameEscaping( ctxHdl IN ctxType,
                                   flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Turn on escaping? `TRUE` for yes, otherwise `FALSE`.

propagateOriginalException()

Description

Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an OracleXMLSQLException.

Syntax

```
PROCEDURE propagateOriginalException( c txHdl IN ctxType,
                                      flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Propagate original exception? `TRUE` for yes, otherwise `FALSE`.

getExceptionContent()

Description

Returns the thrown exception's error code and error message through its arguments (that is, SQL error code). This is to get around the fact that the JVM throws an exception on top of whatever exception was raised; thus, rendering PL/SQL unable to access the original exception.

Syntax

```
PROCEDURE getExceptionContent( ctxHdl IN ctxType,
                               errNo OUT NUMBER,
                               errMsg OUT VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
errNo	(IN)	Error number.
errMsg	(IN	Error message.

getDTD()

Description

Generates and returns the DTD based on the SQL query used to initialize the context. The options are described in the following table.

Syntax	Description
FUNCTION getDTD(ctxHdl IN ctxType, withVer IN BOOLEAN := false) RETURN CLOB;	Function that generates the DTD based on the SQL query used to initialize the context.
PROCEDURE getDTD(ctx IN ctxType, xDoc IN CLOB, withVer IN BOOLEAN := false);	Procedure that generates the DTD based on the SQL query used to initialize the context and xDOC in CLOB.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
withVer	(IN)	Generate the version info? `TRUE` for yes, otherwise `FALSE`.
`xDoc`	(IN)	Clob into which to write the generated XML doc.

getNumRowsProcessed()

Description

Return the number of rows processed for the query.

Syntax

```
FUNCTION getNumRowsProcessed( ctx IN ctxType) RETURN NUMBER;
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

getVersion()

Description

Prints the version of the XSU in use.

Syntax

```
PROCEDURE getVersion();
```

getXML()

Description

Creates the new context, executes the query, gets the XML back and closes the context. This is a convenience function. The context doesn't have to be explicitly opened or closed. The options are described in the following table.

Syntax	Description
FUNCTION getXML(sqlQuery IN VARCHAR2, metaType IN NUMBER := NONE) RETURN CLOB;	This function uses the sqlQuery in string form.
FUNCTION getXML(sqlQuery IN CLOB, metaType IN NUMBER := NONE) RETURN CLOB;	This function uses the sqlQuery in clob form.

FUNCTION getXML(
ctxHdl IN ctxType,
metaType IN NUMBER := NONE);
RETURN CLOB
	This function generates the XML doc. based on the SQL query used to initialize the context.

PROCEDURE getXML(ctxHdl IN ctxType, xDoc IN CLOB, metaType IN NUMBER := NONE);	This procedure generates the XML doc. based on the SQL query used to initialize the context.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
sqlQuery	(IN)	SQLQuery.
metaType	(IN)	XML metadata type (NONE, DTD, or SCHEMA).
sDoc	(IN)	Clob into which to write the generated XML doc.

DBMS_XMLSave Package

Description of DBMS_XMLSave

This API provides XML_to_DB type functionality.

Types of DBMS_XMLSave

Table 20–4 Types of DBMS_XMLSave

Type	Description
ctxType	The type of the query context handle. The type of the query context handle. This the return type of newContext().

Constants of DBMS_XMLSave

Table 20–5 Constants of DBMS_XMLSave

Constant	Description
DEFAULT_ROWTAG	The default tag name for the element corresponding to db. records. -- ROW
DEFAULT_DATE_FORMAT	Default date mask. -- 'MM/dd/yyyy HH:mm:ss'
MATCH_CASE	Used to specify that when mapping XML elements to DB. entities the XSU should be case sensitive.
IGNORE_CASE	Used to specify that when mapping XML elements to DB. entities the XSU should be case insensitive.

Functions and Procedures of DBMS_XMLSave

Table 20–6 Summary of Functions and Procedures of DBMS_XMLSave

Functions/Procedures	Description
newContext() on page 20-22	Creates a save context, and returns the context handle.
closeContext() on page 20-22	It closes / deallocates a particular save context.
setRowTag() on page 20-22	Names the tag used in the XML doc., to enclose the XML elements corresponding to db.
setIgnoreCase() on page 20-23	The XSU does mapping of XML elements to db.

Table 20–6 Summary of Functions and Procedures of DBMS_XMLSave

Functions/Procedures	Description
setDateFormat() on page 20-23	Describes to the XSU the format of the dates in the XML document.
setBatchSize() on page 20-24	Changes the batch size used during DML operations.
setCommitBatch() on page 20-24	Sets the commit batch size.
setSQLToXMLNameEscaping() on page 20-25	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.
setUpdateColumn() on page 20-25	Adds a column to the "update column list".
clearUpdateColumnList() on page 20-26	Clears the update column list.
setPreserveWhitespace() on page 20-26	Tells the XSU whether to preserve whitespace or not.
setKeyColumn() on page 20-26	This methods adds a column to the "key column list".
clearKeyColumnList() on page 20-27	Clears the key column list.
setXSLT() on page 20-27	Registers a XSL transform to be applied to the XML to be saved.
setXSLTParam() on page 20-28	Sets the value of a top-level stylesheet parameter.
removeXSLTParam() on page 20-29	Removes the value of a top-level stylesheet parameter
insertXML() on page 20-29	Inserts the XML document into the table specified at the context creation time.
updateXML() on page 20-30	Updates the table given the XML document.
deleteXML() on page 20-30	Deletes records specified by data from the XML document, from the table specified at the context creation time.
propagateOriginalException() on page 20-31	Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an OracleXMLSQLException.
getExceptionContent() on page 20-31	Returns the thrown exception's error code and error message through its arguments.

newContext()

Description

Creates a save context, and returns the context handle.

Syntax

```
FUNCTION newContext(t argetTable IN VARCHAR2) RETURN ctxType;
```

Parameter	IN / OUT	Description
targetTable	(IN)	The target table into which to load the XML doc.

closeContext()

Description

Closes / deallocates a particular save context

Syntax

```
PROCEDURE closeContext(ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

setRowTag()

Description

Names the tag used in the XML doc., to enclose the XML elements corresponding to db. records.

Syntax

```
PROCEDURE setRowTag( ctxHdl IN ctxType,
                     tag IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
tag	(IN)	Tag name.

setIgnoreCase()

Description

The XSU does mapping of XML elements to db columns / attributes based on the element names (XML tags). This function tells the XSU to do this match case insensitive.

Syntax

```
PROCEDURE setIgnoreCase( ctxHdl IN ctxType,
                         flag IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Ignore tag case in the XML doc? 0=FALSE, 1=TRUE.

setDateFormat()

Description

Describes to the XSU the format of the dates in the XML document. The syntax of the date format pattern (that is, the date mask), should conform to the requirements of the java.text.SimpleDateFormat class. Setting the mask to null or an empty string, results the use of the default mask -- OracleXMLCore.DATE_FORMAT.

Syntax

```
PROCEDURE setDateFormat( ctxHdl IN ctxType,
                         mask IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

Parameter	IN / OUT	Description
mask	(IN)	The date mask.

setBatchSize()

Description

Changes the batch size used during DML operations. When performing inserts, updates or deletes, it is better to batch the operations so that they get executed in one shot rather than as separate statements. The flip side is that more memory is needed to buffer all the bind values. Note that when batching is used, a commit occurs only after a batch is executed. So if one of the statement inside a batch fails, the whole batch is rolled back. This is a small price to pay considering the performance gain; nevertheless, if this behavior is unacceptable, then set the batch size to 1.

Syntax

```
PROCEDURE setBatchSize( ctxHdl IN ctxType,
                        batchSize IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
batchSize	(IN)	Batch size.

setCommitBatch()

Description

Sets the commit batch size. The commit batch size refers to the number or records inserted after which a commit should follow. Note that if commitBatch is < 1 or the session is in "auto-commit" mode then the XSU does not make any explicit commit's. By default the commit-batch size is 0.

Syntax

```
PROCEDURE setCommitBatch( ctxHdl IN ctxType,
                          batchSize IN NUMBER);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
batchSize	(IN)	Commit batch size.

setSQLToXMLNameEscaping()

Description

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

```
PROCEDURE setSQLToXMLNameEscaping( ctxHdl IN ctxType,
                                   flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Turn on escaping?

setUpdateColumn()

Description

Adds a column to the "update column list". In case of insert, the default is to insert values to all the columns in the table; on the other hand, 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 the update column list is specified, the columns making up this list alone will get updated or inserted into.

Syntax

```
PROCEDURE setUpdateColumn( ctxHdl IN ctxType,
                           colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

Parameter	IN / OUT	Description
colName	(IN)	Column to be added to the update column list.

clearUpdateColumnList()

Description

Clears the update column list.

Syntax

```
PROCEDURE clearUpdateColumnList( ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

setPreserveWhitespace()

Description

Tells the XSU whether or not to preserve whitespace.

Syntax

```
PROCEDURE setPreserveWhitespace( ctxHdl IN ctxType,
                                 flag IN BOOLEAN := true);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Should XSU preserve whitespace?

setKeyColumn()

Description

This methods adds a column to the "key column list". In case of update or delete, it is the columns in the key column list that make up the where clause of the update/delete statement. The key columns list must be specified before updates can be done; yet, it is only optional for delete operations.

Syntax

```
PROCEDURE setKeyColumn( ctxHdl IN ctxType,
                        colName IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
colName	(IN)	Column to be added to the key column list.

clearKeyColumnList()

Description

Clears the key column list.

Syntax

```
PROCEDURE clearKeyColumnList( ctxHdl IN ctxType);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.

setXSLT()

Description

Registers an XSL transform to be applied to the XML to be saved. If a stylesheet was already registered, it gets replaced by the new one. To un-register the stylesheet, pass in null for the URI. The options are described in the following table.

Syntax	Description
PROCEDURE setXSLT(ctxHdl IN ctxType, uri IN VARCHAR2, ref IN VARCHAR2 := null);	Passes in the stylesheet through a URI.
PROCEDURE setXSLT(ctxHdl IN ctxType, stylesheet IN CLOB, ref IN VARCHAR2 := null);	Passes in the stylesheet through a CLOB.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
uri	(IN)	URI to the stylesheet to register.
ref	(IN)	URL for include, import, and external entities.
stylesheet	(IN)	CLOB containing the stylesheet to register.

setXSLTParam()

Description

Sets the value of a top-level stylesheet parameter. The parameter is expected to be a valid XPath expression (not that string literal values would therefore have to be explicitly quoted).

Syntax

```
PROCEDURE setXSLTParam( ctxHdl IN ctxType,
                        name IN VARCHAR2,
                        value IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Parameter name.
value	(IN)	Parameter value as an XPath expression

removeXSLTParam()

Description

Removes the value of a top-level stylesheet parameter.

Syntax

```
PROCEDURE removeXSLTParam( ctxHdl IN ctxType,
                           name IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
name	(IN)	Parameter name.

insertXML()

Description

Inserts the XML document into the table specified at the context creation time, and returns the number of rows inserted. The options are described in the following table.

Syntax	Description
FUNCTION insertXML(ctxHdl IN ctxType, xDoc IN VARCHAR2) RETURN NUMBER;	Passes in the `xDoc` parameter as a VARCHAR2.
FUNCTION insertXML(ctxHdl IN ctxType, xDoc IN CLOB) RETURN NUMBER;	Passes in the `xDoc` parameter as a CLOB.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.

updateXML()

Description

Updates the table specified at the context creation time with data from the XML document, and returns the number of rows updated. The options are described in the following table.

Syntax	Description
FUNCTION updateXML(ctxHdl IN ctxType, xDoc IN VARCHAR2) RETURN NUMBER;	Passes in the `xDoc` parameter as a VARCHAR2.
FUNCTION updateXML(ctxHdl IN ctxType, xDoc IN CLOB) RETURN NUMBER;	Passes in the `xDoc` parameter as a CLOB.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.

deleteXML()

Description

Deletes records specified by data from the XML document from the table specified at the context creation time, and returns the number of rows deleted. The options are described in the following table.

Syntax	Description
FUNCTION deleteXML(ctxHdl IN ctxPType, xDoc IN VARCHAR2) RETURN NUMBER;	Uses a VARCHAR2 type for the `xDoc` parameter.

Syntax	Description
FUNCTION deleteXML(ctxHdl IN ctxType, xDoc IN CLOB) RETURN NUMBER;	Uses a CLOB type for the `xDoc` parameter.

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
xDoc	(IN)	String containing the XML document.

propagateOriginalException()

Description

Tells the XSU that if an exception is raised, and is being thrown, the XSU should throw the very exception raised; rather then, wrapping it with an OracleXMLSQLException.

Syntax

```
PROCEDURE propagateOriginalException( ctxHdl IN ctxType,
                                      flag IN BOOLEAN);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
flag	(IN)	Propagate the original exception? 0=FALSE, 1=TRUE.

getExceptionContent()

Description

Through its arguments, this method returns the thrown exception's error code and error message (that is, SQL error code) This is to get around the fact that the JVM throws an exception on top of whatever exception was raised; thus, rendering PL/SQL unable to access the original exception.

Syntax

```
PROCEDURE getExceptionContent( ctxHdl IN ctxType,
                               errNo OUT NUMBER,
                               errMsg OUT VARCHAR2);
```

Parameter	IN / OUT	Description
ctxHdl	(IN)	Context handle.
errNo	(IN)	Error number.
errMsg	(IN)	Error message.

Part V

XML Database Support: Oracle XML DB for Java

Include text to introduce the contents of this book part and provide a list of the chapters in this book part. Insert the list of chapters using cross-references so they are links in HTML.

This part contains the following chapters:

21 Java API for XMLType

The classes described in this chapter are contained in package *oracle.xdb.dom* and implement the Java DOM API for XMLType. In addition to implementing the W3C DOM Recommendation, the DOM API for Oracle XML DB provides Oracle-specific extensions. This chapter contains these sections:

Table 21–1 lists each class and the W3C DOM interface it implements.

Table 21–1 Java DOM API for XMLType Classes and W3C Interfaces

Java DOM API for XMLType Class	W3C DOM Interface Recommendation
XDBAttribute Class	org.w3c.dom.Attribute
XDBCData Class	org.w3c.dom.CData
XDBCharData Class	org.w3c.dom.CharData
XDBComment Class	org.w3c.dom.Comment
XDBDocument Class	org.w3c.dom.Document
XDBDomImplementation Class	org.w3c.dom.DOMImplementation
XDBElement Class	org.w3c.dom.Element
XDBEntity Class	org.w3c.dom.Entity
XDBNamedNodeMap Class	org.w3c.dom.NamedNodeMap
XDBNode Class	org.w3c.dom.Node
XDBNodeList Class	org.w3c.dom.NodeList
XDBNotation Class	org.w3c.dom.Notation
XDBProcInst Class	org.w3c.dom.ProcessingInstruction
XDBText Class	org.w3c.dom.Text

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied Java Packages Reference*

XDBAttribute Class

Description of XDBAttribute

This class implements `org.w3c.dom.Attribute`, the W3C DOM Node interface for interacting with XOBs.

Syntax XDBAttribute

```
public class XDBAttribute
```

`oracle.xdb.dom.XDBAttribute`

XDBCData Class

Description of XDBCData

This class implements `org.w3c.dom.CData`, the W3C text interface.

Syntax of XDBCData

```
public class XDBCData
```

`oracle.xdb.dom.XDBCData`

XDBCharData Class

Description of XDBCharData

This class implements `org.w3c.dom.CharData`, the W3C CharacterData interface.

Syntax of XDBCharData

```
public class XDBCharData
```

`oracle.xdb.dom.XDBCharData`

XDBComment Class

Description of XDBComment

This class implements the `org.w3c.dom.Comment` interface.

Syntax of XDBComment

```
public class XDBComment
```

`oracle.xdb.dom.XDBComment`

XDBDocument Class

Description of XDBDocument

This class implements the `org.w3c.dom.Document` interface.

Syntax of XDBDocument

```
public class XDBDocument
```

`oracle.xdb.dom.XDBDocument`

Methods for XDBDocument

XDBDocument()

Description

Class constructor. The options are described in the following table.

Syntax	Description
public XDBDocument();	Creates new Document. Can be used in server only.
public XDBDocument(byte[] source);	Populates Document from source. Can be used in server only.
public XDBDocument(Connection conn);	Opens connection for caching Document source.

Syntax	Description
public XDBDocument(Connection conn, byte[] source);	Connection for caching bytes for Document source.
public XDBDocument(Connection conn, String source);	Opens connection for caching string containing XML text.
public XDBDocument(String source);	The string containing XML text. Can be used in server only.

Parameter	Description
source	Contains XML text.
conn	Connection to be used.

XDBDomImplementation Class

Description of XDBDomImplementation

This class implements `org.w3c.dom.DomImplementation`.

Syntax of XDBDomImplementation

```
public class XDBDomImplementation
```

`oracle.xdb.dom.XDBDomImplementation`

Methods for XDBDomImplementation

XDBDomImplementation()

Description

Opens a JDBC connection to the server.

Syntax

```
public  XDBDomImplementation();
```

XDBElement Class

Description of XDBElement

This class implements `org.w3c.dom.Element`.

Syntax of XDBElement

```
public class XDBElement
```

oracle.xdb.dom.XDBElement

XDBEntity Class

Description of XDBEntity

This class implements `org.w3c.dom.Entity`.

Syntax of XDBEntity

```
public class XDBEntity
```

oracle.xdb.dom.XDBEntity

XDBNamedNodeMap Class

Description of XDBNamedNodeMap

This class implements `org.w3c.dom.NamedNodeMap`.

Syntax of XDBNamedNodeMap

```
public class XDBNamedNodeMap
```

oracle.xdb.dom.XDBNamedNodeMap

XDBNode Class

Syntax of XDBNode

```
public abstract class XDBNode
```

oracle.xdb.dom.XDBNode

Description of XDBNode

This class implements `org.w3c.dom.Node`, the W3C DOM Node interface for interacting with XOBs.

Methods for XDBNode Class

write()

Description

Writes the XML for this Node (and all subnodes) to an OutputStream. If the OutputStream is a ServletOutputStream, the servlet output is committed and the data is written using a native stream mechanism.

Syntax

```
public void write( OutputStream s,
                   String charEncoding,
                   short indent);
```

Parameter	Description
s	The stream to write the output to. Contains XML text.
charEncoding	The IANA char code, such as “ISO-8859”.
indent	Number of chars to indent nested elements.

XDBNodeList Class

Description of XDBNodeList

This class implements `org.w3c.dom.NodeList`.

Syntax of XDBNodeList

```
public class XDBNodeList
```

`oracle.xdb.dom.XDBNodeList`

XDBNotation Class

Description of XDBNotation

This class implements `org.w3c.dom.Notation.`

Syntax of XDBNotation

```
public class XDBNotation
```

`oracle.xdb.dom.XDBNotation`

XDBProcInst Class

Description of XDBProcInst

This class implements `org.w3c.dom.ProcInst`, the W3C DOM ProcessingInstruction interface.

Syntax of XDBProcInst

```
public class XDBProcInst
```

`oracle.xdb.dom.XDBProcInst`

XDBText Class

Description of XDBText

This class implements `org.w3c.dom.Text`.

Syntax of XDBText

```
public class XDBText
```

`oracle.xdb.dom.XDBText`

XMLType Class

Description of XMLType

XMLType implements the Java methods for the sql type SYS.XMLTYPE.

Syntax of XMLType

```
public class XMLType
```

`oracle.xdb.XMLType`

Methods for XMLType Class

Table 21–2 Summary of Methods for XMLType

Method	Description
createXML() on page 21-10	Creates an XMLType.
getInputStream() on page 21-11	Returns an InputStream corresponding the XMLType data.
getStringVal() on page 21-11	Retrieves the string value containing the XML data from the XMLType.
getClobVal() on page 21-11	Retrieves the CLOB value containing the XML data from the XMLType.
extract() on page 21-12	Extracts the given set of nodes from the XMLType.
existsNode() on page 21-12	Checks for the existence of the given set of nodes in the XMLType.

Table 21–2 Summary of Methods for XMLType (Cont.)

Method	Description
transform() on page 21-13	Transforms the XMLType using the given XSL document.
isFragment() on page 21-13	Checks if the XMLType is a regular document or a document fragment.
getDOM() on page 21-13	Retrieves the DOM document associated with the XMLType.
writeToOutputStream() on page 21-14	Writes the XML Type data to the specified OutputStream.

createXML()

Description

Creates an XMLType. Throws `java.sql.SQLException` if the XMLType could not be created. The options are described in the following table.

Syntax	Description
public static XMLType createXML(OPAQUE opq);	Creates and returns an XMLType given the opaque type containing the XMLType bytes.
public static XMLType createXML(Connection conn, String xmlval);	Creates and returns an XMLType given the string containing the XML data.
public static XMLType createXML(Connection conn, CLOB xmlval);	Creates and returns an XMLType given a CLOB containing the XML data.
public static XMLType createXML(Connection conn, Document domdoc);	Creates and returns an XMLType given an instance of the DOM document.
public static XMLType createXML(Connection conn, InputStream is);	Creates and returns an XMLType given the InputStream for the XML data.

Parameter	Description
opq	The opaque object from which the XMLType is to be constructed.
conn	The connection object to be used.
xmlval	Contains the XML data.
domdoc	The DOM Document which represents the DOM tree.

getInputStream()

Description

Returns an input stream which corresponds to the XMLType document. Users can then read the XML data using the InputStream methods.

Syntax

```
public InputStream getInputStream();
```

getStringVal()

Description

Retrieves the string value containing the XML data from the XMLType. Throws `java.sql.SQLException.`

Syntax

```
public String getStringVal();
```

getClobVal()

Description

Retrieves the CLOB value containing the XML data from the XMLType. Throws `java.sql.SQLException.`

Syntax

```
public CLOB getClobVal();
```

extract()

Description

Extracts and returns the given set of nodes from the XMLType. This set of nodes is specified by the XPath expression. The original XMLType remains unchanged. Works only in the thick case. If no nodes match the specified expression, returns `NULL`. Throws `java.sql.SQLException.`

Syntax

```
public XMLType extract( String xpath,
                        String nsmap);
```

Parameter	Description
xpath	The xpath expression which specifies for which nodes to search.
nsmap	The map of namespaces which resolves the prefixes in the xpath expression; format is “xmlns=a.com xmlns:b=b.com”

existsNode()

Description

Checks for the existence of the given set of nodes in the XMLType. This set of nodes is specified by the xpath expression. Returns `TRUE` if specified nodes exist in the XMLType; otherwise, returns `FALSE`. Throws `java.sql.SQLException.`

Syntax

```
public boolean existsNode( String xpath,
                           String nsmap);
```

Parameter	Description
xpath	The xpath expression which specifies for which nodes to search.
nsmap	The map of namespaces which resolves the prefixes in the xpath expression; format is “xmlns=a.com xmlns:b=b.com”

transform()

Description

Transforms and returns the XMLType using the given XSL document. The new (transformed) XML document is returned. Throws `java.sql.SQLException.`

Syntax

```
public XMLType transform( XMLType xsldoc,
                          String parammap);
```

Parameter	Description
xsldoc	The XSL document to be applied to the XMLType.
parammap	The top level parameters to be passed to the XSL transformation. This should be of the format “a=b c=d e=f”. This can be `NULL`.

isFragment()

Description

Checks if the XMLType is a regular document or a document fragment. Returns `TRUE` if doc is a fragment; otherwise, returns `FALSE`. Throws `java.sql.SQLException.`

Syntax

```
public boolean isFragment();
```

getDOM()

Description

Retrieves the DOM document associated with the XMLType. This document is the org.w3c.dom.Document. The caller can perform all the DOM operations on the Document. If the document is a binary document, the getDOM function will return `NULL`. Throws `java.sql.SQLException.`

Syntax

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

writeToOutputStream()

Description

Writes the XML data into the specified OutputStream.

Syntax

```
public void writeToOutputStream( OutputStream os);
```

Parameter	Description
`os`	Specified Output Stream where data is written.

22

Oracle XML DB API for Java Beans

The Oracle XML DB Java Beans API implements the mapping relationship for Java Beans classes to Java, XML Schema, and SQL entities and data types.

This chapter contains these tables:

- Mapping for XML Schema to Java Beans Classes and Methods
- Mapping for Java Beans API to XML Schema, SQL, and Java Beans Data Types

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied Java Packages Reference*
- http://www.w3.org/XML/Schema

Mapping for XML Schema to Java Beans Classes and Methods

Table 22–1 lists mapping of XML Schema entities to the Java Beans classes and their methods.

Table 22–1 Mapping of XML Schema entities to Java Beans classes and methods

XML Schema Entity	Java Beans Classes/Methods	Description
Attribute	Get/setAttribute{attrname}	Get/set attribute for document child element and its specified attribute.
Complextype	Get/set{complextype classname}.	For complex children, separate bean classes get generated. Extras (processing instructions or comments). DOM classes for processing instructions and comments are returned.
Scalar data	Get/set{scalar type name}	Children with maxoccurs > 1 Get/set List of type of child.

Mapping for Java Beans API to XML Schema, SQL, and Java Beans Data Types

Table 22–2 lists mapping used by the Java Beans API between XML Schema, SQL, and Java data types.

Table 22–2 Mapping From XML Schema, SQL, and Java Data Types

XML Schema Data Type	SQL DataType	Java Beans Data Type
Boolean	boolean	boolean
String	URI reference	ID
IDREF	ENTITY	NOTATION
Language	NCName	Name
java.lang.String	String	DECIMAL
INTEGER	LONG	SHORT
INT	POSITIVEINTEGER	NONPOSITIVEINTEGER
oracle.sql.Number	int	FLOAT

Table 22–2 Mapping From XML Schema, SQL, and Java Data Types (Cont.)

XML Schema Data Type	SQL DataType	Java Beans Data Type
DOUBLE	oracle.sql.Number	float
TIMEDURATION	TIMEPERIOD	RECURRINGDURATION
DATE	TIME	MONTH,YEAR
RECURRINGDATE	java.sql.Timestamp	Time
REF	oracle.sql.Ref	Ref
BINARY	oracle.sql.RAW	Byte[]
QNAME	java.lang.String	String

23

Resource APIs for Java/JNDI

This chapter documents package *oracle.xdb.spi*. The classes contained in *oracle.xdb.spi* implement the service provider drivers that provide the application with common access to JNDI.

The classes contained in *oracle.xdb.spi* implement the service provider drivers that provide the application with common access to Java Naming and Directory Interface (JNDI). JNDI is a programming interface from Sun for connecting Java programs to naming and directory services such as DNS, LDAP and NDS. JNDI (Java Naming and Directory Interface) is a programming interface from Sun for connecting Java programs to naming and directory services such as DNS, LDAP and NDS.

The application is written to the JNDI API, and the directory drivers are written to the JNDI SPI (Service Provider Interface). The classes in *oracle.xdb.spi* implement core JNDI SPI interfaces and WebDAV support for Oracle XML DB.

This chapter contains these sections:

- XDBContext Class
- XDBContextFactory Class
- XDBNameParser Class
- XDBNamingEnumeration Class
- XDBResource Class

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied Java Packages Reference*

XDBContext Class

Description of XDBContext

This class implements the Java naming and context interface for Oracle XML DB, which extends `javax.naming.context`. The current implementation has no federation support, which makes it completely unaware of the existence of other namespaces.

Syntax of XDBContext

```
public class XDBContext
```

`oracle.xdb.spi.XDBContext`

Methods of XDBContext

XDBContext()

Description

Constructor for class XDBContext. The options are described in the following table.

Syntax	Description
public XDBContext(Hashtable env);	Creates an instance of XDBContext class given the environment.
public XDBContext(Hashtable env, String path);	Creates an instance of XDBContext class given the environment and path.

Parameter	Description
env	Environment to describe the properties of context.
path	Initial path for the context.

XDBContextFactory Class

Description of XDBContextFactory

This class implements `javax.naming.context`.

Syntax of XDBContextFactory

```
public class XDBContextFactory
oracle.xdb.spi.XDBContextFactory
```

Methods of XDBContextFactory

XDBContextFactory()

Description

Constructor for class XDBContextFactory.

Syntax

```
public  XDBContextFactory();
```

XDBNameParser Class

Description of XDBNameParser

Implements `javax.naming.NameParser`

Syntax of XDBNameParser

```
public class XDBNameParser
oracle.xdb.spi.XDBNameParser
```

XDBNamingEnumeration Class

Description of XDBNamingEnumeration

Implements `javax.naming.NamingEnumeration`

Syntax of XDBNamingEnumeration

```
public class XDBNamingEnumeration
oracle.xdb.spi.XDBNamingEnumeration
```

XDBResource Class

Description of XDBResource

This class implements the core features for the Oracle XML DB JNDI service provider interface (SPI). The current implementation has no federation support, being completely unaware of the existence of other namespaces.

Syntax of XDBResource

```
public class XDBResource extends java.lang.Object

java.lang.Object
  |
  +--oracle.xdb.spi.XDBResource
```

Methods of XDBResource

Table 23–1 Summary of Methods of XDBResource

Method	Description
XDBResource() on page 23-5	Creates a new instance of XDBResource.
getAuthor() on page 23-5	Returns author of the resource.
getComment() on page 23-6	Returns the DAV comment of the resource.
getContent() on page 23-6	Returns the content of the resource.
getContentType() on page 23-6	Returns the content type of the resource.
getCreateDate() on page 23-6	Returns the create date of the resource.
getDisplayName() on page 23-7	Returns the display name of the resource.
getLanguage() on page 23-7	Returns the language of the resource.
getLastModDate() on page 23-7	Returns the last modification date of the resource.
getOwnerId() on page 23-7	Returns the owner ID of the resource.
setACL() on page 23-8	Sets the ACL on the resource.
setAuthor() on page 23-8	Sets the author of the resource.
setComment() on page 23-8	Sets the DAV comment of the resource.
setContent() on page 23-9	Sets the content of the resource.

Table 23–1 Summary of (Cont.) Methods of XDBResource (Cont.)

Method	Description
setContentType() on page 23-9	Sets the content type of the resource.
setCreateDate() on page 23-9	Sets the creation date of the resource.
setDisplayName() on page 23-10	Sets the display name of the resource.
setLanguage() on page 23-10	Sets the language of the resource.
setLastModDate() on page 23-10	Sets the last modification date of the resource.
setOwnerId() on page 23-11	Sets the owner ID of the resource.

XDBResource()

Description

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

Syntax	Description
public XDBResource(Hashtable env);	Creates a new instance of XDBResource given the environment.
public XDBResource(Hashtable env, String path);	Creates a new instance of XDBResource given the environment and path.

Parameter	Description
env	Environment passed in.
path	Path to the resource

getAuthor()

Description

Retrieves the author of the resource.

Syntax

```
public String getAuthor();
```

getComment()

Description

Retrieves the DAV (Web Distributed Authoring and Versioning) comment of the resource.

Syntax

```
public String getComment();
```

getContent()

Description

Returns the content of the resource.

Syntax

```
public Object getContent();
```

getContentType()

Description

Returns the content type of the resource.

Syntax

```
public String getContentType();
```

getCreateDate()

Description

Returns the creation date of the resource.

Syntax

```
public Date getCreateDate();
```

getDisplayName()

Description

Returns the display name of the resource.

Syntax

```
public String getDisplayName();
```

getLanguage()

Description

Returns the Language of the resource.

Syntax

```
public String getLanguage();
```

getLastModDate()

Description

Returns the last modification date of the resource.

Syntax

```
public Date getLastModDate();
```

getOwnerId()

Description

Returns the owner id of the resource. The owner id value expected by this method is the user id value for the database user as provided by the catalog views such as ALL_USERS, and so on.

Syntax

```
public long getOwnerId();
```

setACL()

Description

Sets the ACL on the resource.

Syntax

```
public void setACL( String aclpath);
```

Parameter	Description
aclpath	The path to the ACL resource.

setAuthor()

Description

Sets the author of the resource.

Syntax

```
public void setAuthor( String authname);
```

Parameter	Description
authname	Author of the resource.

setComment()

Description

Sets the DAV (Web Distributed Authoring and Versioning) comment of the resource.

Syntax

```
public void setComment(String davcom);
```

Parameter	Description
davcom	DAV comment of the resource.

setContent()

Description

Sets the content of the resource.

Syntax

```
public void setContent( Object xmlobj);
```

Parameter	Description
xmlobj	Content of the resource.

setContentType()

Description

Sets the content type of the resource.

Syntax

```
public void setContentType( String conttype);
```

Parameter	Description
conttype	Content type of the resource.

setCreateDate()

Description

Sets the creation date of the resource.

Syntax

```
public void setCreateDate( Date credate);
```

Parameter	Description
credate	Creation date of the resource.

setDisplayName()

Description

Sets the display name of the resource

Syntax

```
public void setDisplayName( String dname);
```

Parameter	Description
dname	Display name of the resource.

setLanguage()

Description

Sets the language of the resource.

Syntax

```
public void setLanguage(String lang);
```

Parameter	Description
lang	Language of the resource.

setLastModDate()

Description

Sets the last modification date of the resource.

Syntax

```
public void setLastModDate( Date d);
```

Parameter	Description
d	Last modification date of the resource.

setOwnerId()

Description

Sets the owner id of the resource. The owner id value expected by this method is the user id value for the database user as provided by the catalog views such as ALL_USERS, and so on.

Syntax

```
public void setOwnerId( long ownerid);
```

Parameter	Description
ownerid	Owner id of the resource.

Part VI

XML Database Support: Oracle XML DB for PL/SQL

This section contains the following chapters:

24

XMLType API for PL/SQL

This chapter describes the types and functions used for native XML support in the server, namely:

- Description of XMLType
- Functions and Procedures of XMLType

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*

XMLType

Description of XMLType

XMLType is a system-defined opaque type for handling XML data. XMLType has predefined member functions on it to extract XML nodes and fragments.

You can create columns of XMLType and insert XML documents into it. You can also generate XML documents as XMLType instances dynamically using the SYS_XMLGEN and SYS_XMLAGG SQL functions.

For example,

```
CREATE TABLE Xml_tab ( xmlval xmltype);

INSERT INTO Xml_tab VALUES (
   xmltype('<?xml version="1.0"?>
               <EMP>
                  <EMPNO>221</EMPNO>
                  <ENAME>John</ENAME>
               </EMP>'));

INSERT INTO Xml_tab VALUES (
   xmltype('<?xml version="1.0"?>
               <PO>
                  <PONO>331</PONO>
                  <PONAME>PO_1</PONAME>
               </PO>'));
```

-- now extract the numerical values for the employee numbers

```
SELECT e.xmlval.extract('//EMPNO/text()').getNumVal() as empno
   FROM Xml_tab
   WHERE e.xmlval.existsnode('/EMP/EMPNO')  = 1;
```

Functions and Procedures of XMLType

Table 24–1 Summary of Functions and Procedures of XMLType

Function	Description
XMLType() on page 24-4	Constructor that constructs an instance of the XMLType datatype. The constructor can take in the XML as a CLOB, VARCHAR2 or take in a object type.
createXML() on page 5	Static function for creating and returning an XMLType instance.
existsNode() on page 24-7	Takes a XMLType instance and a XPath and returns 1 or 0 indicating if applying the XPath returns a non-empty set of nodes.
extract() on page 24-7	Takes a XMLType instance and an XPath, applies the XPath expression and returns the results as an XMLType.
isFragment() on page 24-8	Checks if the input XMLType instance is a fragment or not. A fragment is a XML instance, which has more than one root element.
getClobVal() on page 24-8	Returns the value of the XMLtype instance as a CLOB
getNumVal() on page 24-8	Returns the value of the XMLtype instance as a NUMBER. This is only valid if the input XMLtpye instance contains a simple text node and is convertible to a number.
getStringVal() on page 24-9	Returns the value of the XMLType instance as a string.
transform() on page 24-9	Takes an XMLtype instance and an associated stylesheet (which is also an XMLtype instance) , applies the stylesheet and returns the result as XML.
toObject() on page 24-9	Converts the XMLType instance to an object type.
isSchemaBased() on page 24-10	Returns 1 or 0 indicating if the input XMLType instance is a schema based one or not.
getSchemaURL() on page 24-10	Returns the XML schema URL if the input is a XMLSchema based.
getRootElement() on page 24-11	Returns the root element of the input instance. Returns NULL if the instance is a fragment
createSchemaBasedXML() on page 24-11	Creates a schema based XMLtype instance from the non-schema based instance using the input schema URL.
createNonSchemaBasedXML( ) on page 24-11	Creates a non schema based XML from the input schema based instance.

Table 24–1 Summary of Functions and Procedures of XMLType (Cont.)

Function	Description
getNamespace() on page 24-12	Returns the namespace for the top level element in a schema based document.
schemaValidate() on page 24-12	Validates the input instance according to the XMLSchema. Raises error if the input instance is non-schema based.
isSchemaValidated() on page 24-12	Checks if the instance has been validated against the schema.
setSchemaValidated() on page 24-12	Sets the schema valid flag to avoid costly schema validation.
isSchemaValid() on page 24-13	Checks if the input instance is schema valid according to the given schema URL.

XMLType()

Description

XMLType constructor. The options are described in the following table.

Syntax	Description
constructor function XMLType(xmlData IN clob, schema IN varchar2 := NULL, validated IN number := 0, wellformed IN Number := 0) return self as result	This constructor function creates an optionally schema-based XMLType instance using the specified schema and xml data prameters.
constructor function XMLType(xmlData IN varchar2, schema IN varchar2 := NULL, validated IN number := 0, wellformed IN number := 0) return self as result	This constructor function creates an optionally schema-based XMLType instance using the specified schema and xml data prameters.
constructor function XMLType (xmlData IN "<ADT_1>", schema IN varchar2 := NULL, element IN varchar2 := NULL, validated IN number := 0) return self as result	This constructor function creates an optionally schema-based XMLType instance from the specified object type parameter.

Syntax	Description
onstructor function XMLType(xmlData IN SYS_REFCURSOR, schema in varchar2 := NULL, element in varchar2 := NULL, validated in number := 0) return self as result	This constructor function creates an optionally schema-based XMLType instance from the specified REF CURSOR parameter.

Parameter	IN / OUT	Description
xmlData	(IN)	The actual data in the form of a CLOB, REF cursor, VARCHAR2 or object type.
schema	(IN)	Optional Schema URL to be used to make the input conform to the given schema.
validated	(IN)	Flag to indicate that the instance is valid according to the given XMLSchema. (default 0)
wellformed	(IN)	Flag to indicate that the input is wellformed. If set, then the database would not do well formed check on the input instance. (default 0)
element	(IN)	Optional element name in the case of the ADT_1 or REF CURSOR constructors. (default null)

createXML()

Description

Static function for creating and returning an XMLType instance. The string and clob parameters used to pass in the date must contain well-formed and valid XML documents. The options are described in the following table.

Syntax	Description
STATIC FUNCTION createXML(xmlval IN varchar2) RETURN XMLType deterministic	Creates the XMLType instance from a string.
STATIC FUNCTION createXML(xmlval IN clob) RETURN XMLType	Creates the XMLType instance from a CLOB.

Syntax	Description
STATIC FUNCTION createXML (xmlData IN clob, schema IN varchar2, validated IN number := 0, wellformed IN number := 0) RETURN XMLType deterministic	This static function creates a schema-based XMLType instance using the specified schema and xml data prameters.
STATIC FUNCTION createXML (xmlData IN varchar2, schema IN varchar2, validated IN number := 0, wellformed IN number := 0) RETURN XMLType deterministic	This static function creates a schema-based XMLType instance using the specified schema and xml data prameters.
STATIC FUNCTION createXML (xmlData IN "<ADT_1>", schema IN varchar2 := NULL, element IN varchar2 := NULL, validated IN NUMBER := 0) RETURN XMLType deterministic	Creates an XML instance from an instance of an user-defined type.
STATIC FUNCTION createXML (xmlData IN SYS_REFCURSOR, schema in varchar2 := NULL, element in varchar2 := NULL, validated in number := 0) RETURN XMLType deterministic	Creates an XML instance from a cursor reference. You can pass in any arbitrary SQL query as a CURSOR.

Parameter	IN / OUT	Description
xmlData	(IN)	The actual data in the form of a CLOB, REF cursor, VARCHAR2 or object type.
schema	(IN)	Optional Schema URL to be used to make the input conform to the given schema.
validated	(IN)	Flag to indicate that the instance is valid according to the given XMLSchema. (default 0)
wellformed	(IN)	Flag to indicate that the input is wellformed. If set, then the database would not do well formed check on the input instance. (default 0)
element	(IN)	Optional element name in the case of the ADT_1 or REF CURSOR constructors. (default null)

existsNode()

Description

Member function. Checks if the node exists. If the XPath string is NULL or the document is empty, then a value of 0 is returned, otherwise returns 1. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION existsNode(xpath IN varchar2) RETURN number deterministic	Given an XPath expression, checks if the XPath applied over the document can return any valid nodes.
MEMBER FUNCTION existsNode(xpath in varchar2, nsmap in varchar2) RETURN number deterministic	This member function uses the XPath expression with the namespace information and checks if applying the XPath returns any nodes or not.

Parameter	IN / OUT	Description
xpath	(IN)	The XPath expression to test.
nsmap	(IN)	Optional namespace mapping.

extract()

Description

Member function. Extracts an XMLType fragment and returns an XMLType instance containing the result node(s). If the XPath does not result in any nodes, then returns NULL. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION extract(xpath IN varchar2) RETURN XMLType deterministic	Given an XPath expression, applies the XPath to the document and returns the fragment as an XMLType.
MEMBER FUNCTION extract(xpath IN varchar2, nsmap IN varchar2) RETURN XMLType deterministic	This member function applies the XPath expression along with the namespace mapping, over the XML data to return a XMLType instance containing the resultant fragment.

Parameter	IN / OUT	Description
xpath	(IN)	The XPath expression to apply.
nsmap	(IN)	Optional prefix to namespace mapping information.

isFragment()

Description

Determines if the `XMLType` instance corresponds to a well-formed document, or a fragment. Returns 1 or 0 indicating if the `XMLType` instance contains a fragment or a well-formed document. Returns 1 or 0 indicating if the XMLType instance contains a fragment or a well-formed document.

Syntax

```
MEMBER FUNCTION isFragment() RETURN number deterministic
```

getClobVal()

Member function. Returns a CLOB containing the seralized XML representation; if the returns is a temporary CLOB, then it must be freed after use. The options are described in the following table.

Syntax

```
MEMBER FUNCTION getClobVal() RETURN clob deterministic
```

getNumVal()

Description

Member function. Returns a numeric value, formatted from the text value pointed to by the XMLType instance. The XMLType must point to a valid text node that contains a numerical value. The options are described in the following table.

Syntax

```
MEMBER FUNCTION getNumVal() RETURN number deterministic
```

getStringVal()

Description

Member function. Returns the document as a string. Returns s string containing the seralized XML representation, or in case of text nodes, the text itself. If the XML document is bigger than the maximum size of the varchar2, which is 4000, then an error is raised at run time.

Syntax

```
MEMBER FUNCTION getStringVal() RETURN varchar2 deterministic
```

transform()

Description

Member function. This member function transforms the XML data using the XSL stylesheet argument and the top-level parameters passed as a string of name=value pairs. If any of the arguments other than the parammap is NULL, then a NULL is returned.

Syntax

```
MEMBER FUNCTION transform(xsl IN XMLType, parammap in varchar2 := NULL) RETURN
XMLType deterministic
```

Parameter	IN / OUT	Description
xsl	(IN)	The XSL stylesheet describing the transformation
parammap	(IN)	Top level parameters to the XSL - string of name=value pairs

toObject()

Description

Member procedure. This member procedure converts the XML data into an instance of a user defined type, using the optional schema and top-level element arguments.

Syntax

```
MEMBER PROCEDURE toObject(SELF in XMLType, object OUT "<ADT_1>", schema in
varchar2 := NULL, element in varchar2 := NULL)
```

Parameter	IN / OUT	Description
SELF	(IN)	The instance to be converted. This is implicit if you use it as a member procedure.
object	(IN)	the converted object. An object instance of the required type may be passed in to this function
schema	(IN)	the schema URL. The mapping of the XMLType instance to the converted object instance may be specified using a schema.
element	(IN)	.the top-level element name. An XMLSchema document does not specify the top-level element for a conforming XML instance document. The top-level element name in the XMLSchema document to map the XMLType instance is specified with this parameter.

isSchemaBased()

Description

Member function. Determines whether the XMLType instance is schema-based or not. Returns 1 or 0 depending on whether the XMLType instance is schema-based or not.

Syntax

```
MEMBER FUNCTION isSchemaBased return number deterministic
```

getSchemaURL()

Description

Member function. Returns the XMLSchema URL corresponding to the XMLType instance, if the XMLType instance is a schema-based document. Otherwise returns NULL.

Syntax

```
MEMBER FUNCTION getSchemaURL return varchar2 deterministic
```

getRootElement()

Description

Member function. Gets the root element of the XMLType instance. Returns NULL if the instance is a fragment.

Syntax

```
MEMBER FUNCTION getRootElement return varchar2 deterministic
```

createSchemaBasedXML()

Description

Member function. Creates a schema based XMLType instances from a non-schema based XML and a schemaURL.

Syntax

```
MEMBER FUNCTION createSchemaBasedXML(schema IN varchar2 := NULL) return
sys.XMLType deterministic
```

Parameter	IN / OUT	Description
schema	(IN)	the schema URL If this parameter is NULL, then the XMLType instance must contain a schema URL.

createNonSchemaBasedXML()

Description

Member function. Creates a non schema based XML document from a schema based instance.

Syntax

```
MEMBER FUNCTION createNonSchemaBasedXML return XMLType deterministic
```

getNamespace()

Description

Member function. Returns the namespace of the top level element in the instance. Returns NULL if the input is a fragment or is a non-schema based instance.

Syntax

```
MEMBER FUNCTION getNamespace return varchar2 deterministic
```

schemaValidate()

Description

Member procedure. Validates the XML instance against its schema if it hasn't already been done so. For non-schema based documents an error is raised. If validation fails an error is raised; else, the document’s status is changed to validated.

Syntax

```
MEMBER PROCEDURE schemaValidate
```

isSchemaValidated()

Description

Member function. Returns the validation status of the XMLType instance -- tells if a schema based instance has been actually validated against its schema. Returns 1 if the instance has been validated against the schema, 0 otherwise.

Syntax

```
MEMBER FUNCTION isSchemaValidated return NUMBER deterministic
```

setSchemaValidated()

Description

Memebr function. Sets the VALIDATION state of the input XML instance.

Syntax

```
MEMBER PROCEDURE setSchemaValidated(flag IN BINARY_INTEGER := 1)
```

Parameter	IN / OUT	Description
flag	(IN)	0 - NOT VALIDATED; 1 - VALIDATED; The default value for this parameter is 1.

isSchemaValid()

Description

Memebr function. Checks if the input instance is conformant to a specified schema. Does not change the validation status of the XML instance. If a XML Schema URL is not specified and the xml document is schema based, the conformance is checked against the XMLType instance's own schema.

Syntax

```
member function isSchemaValid(schurl IN VARCHAR2 := NULL, elem IN VARCHAR2 :=
NULL) return NUMBER deterministic
```

Parameter	IN / OUT	Description
schurl	(IN)	The URL of the XML Schema against which to check conformance.
elem	(IN)	Element of a specified schema, against which to validate. This is useful when we have a XML Schema which defines more than one top level element, and we want to check conformance against a specific one of these elements.

25

PL/SQL DOM API for XMLType

The Extensible Markup Language (XML) Parser for PL/SQL, found in the DBMS_XMLDOM Package, is used to access `XMLType` objects and both schema-based and nonschema-based documents.

This chapter details the following:

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

DBMS_XMLDOM Package

Description of DBMS_XMLDOM

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. XML is increasingly 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 Document Object Model, 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 Document Object Model, 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 Document Object Model 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 this PL/SQL adaptation, some changes had to be made:

- Various DOM interfaces such as Node, Element, and so on, have equivalent PL/SQL types DOMNode, DOMElement, and so on, respectively.
- Various DOMException codes such as WRONG_DOCUMENT_ERR, HIERARCHY_REQUEST_ERR, and so on, have similarly named PL/SQL exceptions
- Various DOM Node type codes such as ELEMENT_NODE, ATTRIBUTE_NODE, and so on, have similarly named PL/SQL constants
- Methods defined on a DOM type become functions or procedures that accept it as a parameter. For example, to perform appendChild on a DOM Node n, the `appendChild()` PL/SQL function on page 25-24 is provided,

```
FUNCTION appendChild( n DOMNode,
                      newChild IN DOMNode)
                      RETURN DOMNode;
```

and to perform setAttribute on a DOM Element elem, the setAttribute() PL/SQL procedure on page 25-48 is provided:

```
PROCEDURE setAttribute( elem DOMElement,
                        name IN VARCHAR2,
                        value IN VARCHAR2);
```

DOM defines an inheritance hierarchy. For example, Document, Element, and Attr are defined to be subtypes of Node. Thus, a method defined in the Node interface should be available in these as well. Since, such inheritance is not directly possible in PL/SQL, the makeNode functions need to be invoked on different DOM types to convert these into a DOMNode. The appropriate functions or procedures that accept DOMNodes can then be called to operate on these types. If, subsequently, type specific functionality is desired, the DOMNode can be converted back into the type by using the `make*()` functions, where DOM* is the desired DOM type.

The implementation of this PL/SQL DOM interface followed the DOM standard of revision REC-DOM-Level-1-19981001. The types and methods described in this document are made available by the PL/SQL package `DBMS_XMLDOM`.

- Before database startup, the read-from and write-to directories in the `initialization.ORA` file must be specified; for example:

  ```
  UTL_FILE_DIR=/mypath/insidemypath
  ```

- The read-from and write-to files must be on the server file system.

Types of DBMS_XMLDOM

The following types are defined for DBMS_XMLDOM.DOMTYPE:

Table 25–1 DBMS_XMLDOM Types

Type	Description
DOMNode	Implements the DOM Node interface.
DOMNamedNodeMap	Implements the DOM NamedNodeMap interface.
DOMNodeList	Implements the DOM NodeList interface.
DOMAttr	Implements the DOM Attribute interface.
DOMCDataSection	Implements the DOM CDataSection interface.
DOMCharacterData	Implements the DOM Character Data interface.
DOMComment	Implements the DOM Comment interface.

Table 25–1 DBMS_XMLDOM Types (Cont.)

Type	Description
DOMDocumentFragment	Implements the DOM DocumentFragment interface.
DOMElement	Implements the DOM Element interface.
DOMEntity	Implements the DOM Entity interface.
DOMEntityReference	Implements the DOM EntityReference interface.
DOMNotation	Implements the DOM Notation interface.
DOMProcessingInstruction	Implements the DOM Processing instruction interface.
DOMText	Implements the DOM Text interface.
DOMImplementation	Implements the DOM DOMImplementation interface.
DOMDocumentType	Implements the DOM Document Type interface.
DOMDocument	Implements the DOM Document interface.

Defined Constants of DBMS_XMLDOM

The constants listed in Table 25–2 are defined for `DBMS_XMLDOM`. For example, when a request such as `getNodeType(myNode)` is made, the returned type will be one of the following constants.

Table 25–2 Defined Constants for DBMS_XMLDDOM

Constant	Description
ELEMENT_NODE	The Node is an Element.
ATTRIBUTE_NODE	The Node is an Attribute.
TEXT_NODE	The Node is a Text node.
CDATA_SECTION_NODE	The Node is a CDataSection.
ENTITY_REFERENCE_NODE	The Node is an Entity Reference.
ENTITY_NODE	The Node is an Entity.
PROCESSING_INSTRUCTION_NODE	The Node is a Processing Instruction.
COMMENT_NODE	The Node is a Comment.
DOCUMENT_NODE	The Node is a Document.
DOCUMENT_TYPE_NODE	The Node is a Document Type Definition.
DOCUMENT_FRAGMENT_NODE	The Node is a Document fragment.

Table 25–2 Defined Constants for DBMS_XMLDDOM

Constant	Description
NOTATION_NODE	The Node is a Notation.

Exceptions of DBMS_XMLDOM

The exceptions listed in Table 25–3 are defined for DBMS_XMLDOM:

Table 25–3 Exceptions for DBMS_XMLDDOM

Exception	Description
INDEX_SIZE_ERR	If index or size is negative, or greater than the allowed value.
DOMSTRING_SIZE_ERR	If the specified range of text does not fit into a DOMString.
HIERARCHY_REQUEST_ERR	If any node is inserted somewhere it doesn't belong.
WRONG_DOCUMENT_ERR	If a node is used in a different document than the one that created it (that doesn't support it).
INVALID_CHARACTER_ERR	If an invalid or illegal character is specified, such as in a name. See production 2 in the XML specification for the definition of a legal character, and production 5 for the definition of a legal name character.
NO_DATA_ALLOWED_ERROR	If data is specified for a node which does not support data.
NO_MODIFICATION_ALLOWED_ERR	If an attempt is made to modify an object where modifications are not allowed.
NO_FOUND_ERR	If an attempt is made to reference a node in a context where it does not exist.
NOT_SUPPORTED_ERR	If the implementation does not support the requested type of object or operation.
INUSE_ATTRIBUTE_ERR	If an attempt is made to add an attribute that is already in use elsewhere.

Methods of DBMS_XMLDOM

DBMS_XMLDOM subprograms are divided into groups according to w3c Interfaces.

Table 25–4 Summary of Methods of DBMS_XMLDOM

Group/Method	Description
DOM Node Methods	
isNull() on page 25-12	Tests if the node is `NULL`.
makeAttr() on page 25-12	Casts the node to an Attribute.
makeCDataSection() on page 25-13	Casts the node to a CDataSection.
makeCharacterData() on page 25-13	Casts the node to CharacterData.
makeComment() on page 25-13	Casts the node to a Comment.
makeDocumentFragment() on page 25-14	Casts the node to a DocumentFragment.
makeDocumentType() on page 25-14	Casts the node to a Document Type.
makeElement() on page 25-14	Casts the node to an Element.
makeEntity() on page 25-15	Casts the node to an Entity.
makeEntityReference() on page 25-15	Casts the node to an EntityReference.
makeNotation() on page 25-15	Casts the node to a Notation.
makeProcessingInstruction() on page 25-16	Casts the node to a DOMProcessingInstruction.
makeText() on page 25-16	Casts the node to a DOMText.
makeDocument() on page 25-16	Casts the node to a DOMDocument.
writeToFile() on page 25-17	Writes the contents of the node to a file.
writeToBuffer() on page 25-17	Writes the contents of the node to a buffer.
writeToClob() on page 25-18	Writes the contents of the node to a clob.
getNodeName() on page 25-18	Retrieves the Name of the Node.
getNodeValue() on page 25-19	Retrieves the Value of the Node.
setNodeValue() on page 25-19	Sets the Value of the Node.
getNodeType() on page 25-19	Retrieves the Type of the node.
getParentNode() on page 25-20	Retrieves the parent of the node.
getChildNodes() on page 25-20	Retrieves the children of the node.
getFirstChild() on page 25-20	Retrieves the first child of the node.
getLastChild() on page 25-21	Retrieves the last child of the node.
getPreviousSibling() on page 25-21	Retrieves the previous sibling of the node.

Table 25–4 Summary of Methods of DBMS_XMLDOM (Cont.)

Group/Method	Description
getNextSibling() on page 25-21	Retrieves the next sibling of the node.
getAttributes() on page 25-22	Retrieves the attributes of the node.
getOwnerDocument() on page 25-22	Retrieves the owner document of the node.
insertBefore() on page 25-22	Inserts a child before the reference child.
replaceChild() on page 25-23	Replaces the old child with a new child.
removeChild() on page 25-24	Removes a specified child from a node.
appendChild() on page 25-24	Appends a new child to the node.
hasChildNodes() on page 25-24	Tests if the node has child nodes.
cloneNode() on page 25-25	Clones the node.
DOM Named Node Map methods	
isNull() on page 25-25	Tests if the NamedNodeMap is `NULL`.
getNamedItem() on page 25-26	Retrieves the item specified by the name.
setNamedItem() on page 25-26	Sets the item in the map specified by the name.
removeNamedItem() on page 25-27	Removes the item specified by name.
item() on page 25-27	Retrieves the item given the index in the map.
getLength() on page 25-28	Retrieves the number of items in the map.
DOM Node List Methods	
isNull() on page 25-28	Tests if the Nodelist is `NULL`.
item() on page 25-28	Retrieves the item given the index in the nodelist.
getLength() on page 25-29	Retrieves the number of items in the list.
DOM Attr Methods	
isNull() on page 25-29	Tests if the Attribute Node is `NULL`.
makeNode() on page 25-29	Casts the Attribute to a node.
getQualifiedName() on page 25-30	Retrieves the Qualified Name of the attribute.
getNamespace() on page 25-30	Retrieves the NS URI of the attribute.
getLocalName() on page 25-30	Retrieves the local name of the attribute.
getExpandedName() on page 25-31	Retrieves the expanded name of the attribute.

Table 25–4 Summary of Methods of DBMS_XMLDOM (Cont.)

Group/Method	Description
getName() on page 25-31	Retrieves the name of the attribute.
getSpecified() on page 25-31	Tests if attribute was specified in the owning element.
getValue() on page 25-32	Retrieves the value of the attribute.
setValue() on page 25-32	Sets the value of the attribute.
DOM C Data Section Methods	
isNull() on page 25-32	Tests if the CDataSection is `NULL`.
makeNode() on page 25-33	Casts the CDatasection to a node.
DOM Character Data Methods	
isNull() on page 25-33	Tests if the CharacterData is `NULL`.
makeNode() on page 25-33	Casts the CharacterData to a node.
getData() on page 25-34	Retrieves the data of the node.
setData() on page 25-34	Sets the data to the node.
getLength() on page 25-34	Retrieves the length of the data.
substringData() on page 25-35	Retrieves the substring of the data.
appendData() on page 25-35	Appends the given data to the node data.
insertData() on page 25-35	Inserts the data in the node at the given offSets.
deleteData() on page 25-36	Deletes the data from the given offSets.
replaceData() on page 25-36	Replaces the data from the given offSets.
DOM Comment Methods	
isNull() on page 25-37	Tests if the comment is `NULL`.
makeNode() on page 25-37	Casts the Comment to a node.
DOM Implementation Methods	
isNull() on page 25-38	Tests if the DOMImplementation node is `NULL`.
hasFeature() on page 25-38	Tests if the DOMImplementation implements a given feature.
DOM Document Fragment Methods	
isNull() on page 25-38	Tests if the DocumentFragment is `NULL`.

Table 25–4 Summary of Methods of DBMS_XMLDOM (Cont.)

Group/Method	Description
makeNode() on page 25-39	Casts the Document Fragment to a node.
DOM Document Type Methods	
isNull() on page 25-39	Tests if the Document Type is `NULL`.
makeNode() on page 25-39	Casts the document type to a node.
findEntity() on page 25-40	Finds the specified entity in the document type.
findNotation() on page 25-40	Finds the specified notation in the document type.
getPublicId() on page 25-41	Retrieves the public ID of the document type.
getSystemId() on page 25-41	Retrieves the system ID of the document type.
writeExternalDTDToFile() on page 25-41	Writes the document type definition to a file.
writeExternalDTDToBuffer() on page 25-42	Writes the document type definition to a buffer.
writeExternalDTDToClob() on page 25-42	Writes the document type definition to a clob.
getName() on page 25-43	Retrieves the name of the Document type.
getEntities() on page 25-43	Retrieves the nodemap of entities in the Document type.
getNotations() on page 25-44	Retrieves the nodemap of the notations in the Document type.
DOM Element Methods	
isNull() on page 25-44	Tests if the Element is `NULL`.
makeNode() on page 25-44	Casts the Element to a node.
getQualifiedName() on page 25-45	Retrieves the qualified name of the element.
getNamespace() on page 25-45	Retrieves the NS URI of the element.
getLocalName() on page 25-45	Retrieves the local name of the element.
getExpandedName() on page 25-46	Retrieves the expanded name of the element.
getChildrenByTagName() on page 25-46	Retrieves the children of the element by tag name.
getElementsByTagName() on page 25-47	Retrieves the elements in the subtree by tagname.
resolveNamespacePrefix() on page 25-47	Resolve the prefix to a namespace uri.
getTagName() on page 25-48	Retrieves the Tag name of the element.
getAttribute() on page 25-48	Retrieves the attribute node specified by the name.

Table 25–4 Summary of Methods of DBMS_XMLDOM (Cont.)

Group/Method	Description
setAttribute() on page 25-48	Sets the attribute specified by the name.
removeAttribute() on page 25-49	Removes the attribute specified by the name.
getAttributeNode() on page 25-49	Retrieves the attribute node specified by the name.
setAttributeNode() on page 25-49	Sets the attribute node in the element.
removeAttributeNode() on page 25-50	Removes the attribute node in the element.
normalize() on page 25-50	Normalizes the text children of the element.
DOM Entity Methods	
isNull() on page 25-51	Tests if the Entity is NULL.
makeNode() on page 25-51	Casts the Entity to a node.
getPublicId() on page 25-51	Retrieves the public Id of the entity.
getSystemId() on page 25-52	Retrieves the system Id of the entity.
getNotationName() on page 25-52	Retrieves the notation name of the entity.
DOM Entity Reference Methods	
isNull() on page 25-52	Tests if the Entity Reference is NULL.
makeNode() on page 25-53	Casts the Entity Reference to NULL.
DOM Notation Methods	
isNull() on page 25-53	Tests if the Notation is NULL.
makeNode() on page 25-53	Casts the notation to a node.
getPublicId() on page 25-54	Retrieves the public Id of the notation.
getSystemId() on page 25-54	Retrieves the system Id of the notation.
DOM Processing Instruction Methods	
isNull() on page 25-54	Tests if the Processing Instruction is NULL.
makeNode() on page 25-55	Casts the Processing Instruction to a node.
getData() on page 25-55	Retrieves the data of the processing instruction.
getTarget() on page 25-55	Retrieves the target of the processing instruction.
setData() on page 25-55	Sets the data of the processing instruction.
DOM Text Methods	

Table 25–4 Summary of Methods of DBMS_XMLDOM (Cont.)

Group/Method	Description
isNull() on page 25-56	Tests if the text is NULL.
makeNode() on page 25-56	Casts the text to a node.
splitText() on page 25-57	Splits the contents of the text node into 2 text nodes.
DOM Document Methods	
isNull() on page 25-57	Tests if the document is NULL.
makeNode() on page 25-57	Casts the document to a node.
newDOMDocument() on page 25-58	Creates a new Document.
freeDocument() on page 25-58	Frees the document.
getVersion() on page 25-58	Retrieves the version of the document.
setVersion() on page 25-58	Sets the version of the document.
getCharset() on page 25-59	Retrieves the Character set of the document.
setCharset() on page 25-59	Sets the Character set of the document.
getStandalone() on page 25-59	Retrieves if the document is specified as standalone.
setStandalone() on page 25-60	Sets the document standalone.
writeToFile() on page 25-60	Writes the document to a file.
writeToBuffer() on page 25-61	Writes the document to a buffer.
writeToClob() on page 25-61	Writes the document to a clob.
writeExternalDTDToFile() on page 25-62	Writes the DTD of the document to a file.
writeExternalDTDToBuffer() on page 25-63	Writes the DTD of the document to a buffer.
writeExternalDTDToClob() on page 25-63	Writes the DTD of the document to a clob.
getDoctype() on page 25-64	Retrieves the DTD of the document.
getImplementation() on page 25-64	Retrieves the DOM implementation.
getDocumentElement() on page 25-65	Retrieves the root element of the document.
createElement() on page 25-65	Creates a new Element.
createDocumentFragment() on page 25-65	Creates a new Document Fragment.
createTextNode() on page 25-66	Creates a Text node.
createComment() on page 25-66	Creates a Comment node.

Table 25–4 Summary of Methods of DBMS_XMLDOM (Cont.)

Group/Method	Description
createCDATASection() on page 25-66	Creates a CDatasection node.
createProcessingInstruction() on page 25-67	Creates a Processing Instruction.
createAttribute() on page 25-67	Creates an Attribute.
createEntityReference() on page 25-68	Creates an Entity reference.
getElementsByTagName() on page 25-68	Retrieves the elements in the by tag name.

DOM Node Methods

isNull()

Description

Checks if the given DOMNode is NULL. Returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( n DOMNode) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to check.

makeAttr()

Description

Casts a given DOMNode to a DOMAttr, and returns the DOMAttr.

Syntax

```
FUNCTION makeAttr( n DOMNode) RETURN DOMAttr;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeCDataSection()

Description

Casts a given DOMNode to a DOMCDataSection, and returns the DOMCDataSection.

Syntax

```
FUNCTION makeCDataSection( n DOMNode) RETURN DOMCDataSection;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeCharacterData()

Description

Casts a given DOMNode to a DOMCharacterData, and returns the DOMCharacterData.

Syntax

```
FUNCTION makeCharacterData( n DOMNode) RETURN DOMCharacterData;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeComment()

Description

Casts a given DOMNode to a DOMComment, and returns the DOMComment.

Syntax

```
FUNCTION makeComment(n DOMNode) RETURN DOMComment;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeDocumentFragment()

Description
Casts a given DOMNode to a DOMDocumentFragment, and returns the DOMDocumentFragment.

Syntax
```
FUNCTION makeDocumentFragment( n DOMNode) RETURN DOMDocumentFragment;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeDocumentType()

Description
Casts a given DOMNode to a DOMDocumentType and returns the DOMDocumentType.

Syntax
```
FUNCTION makeDocumentType(n DOMNode) RETURN DOMDocumentType;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeElement()

Description
Casts a given DOMNode to a DOMElement, and returns the DOMElement.

Syntax
```
FUNCTION makeElement(n DOMNode) RETURN DOMElement;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeEntity()

Description

Casts a given DOMNode to a DOMEntity, and returns the DOMEntity.

Syntax

```
FUNCTION makeEntity(n DOMNode) RETURN DOMEntity;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeEntityReference()

Description

Casts a given DOMNode to a DOMEntityReference, and returns the DOMEntityReference.

Syntax

```
FUNCTION makeEntityReference(n DOMNode) RETURN DOMEntityReference;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeNotation()

Description

Casts a given DOMNode to a DOMNotation, and returns the DOMNotation.

Syntax

```
FUNCTION makeNotation(n DOMNode) RETURN DOMNotation;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeProcessingInstruction()

Description

Casts a given DOMNode to a DOMProcessingInstruction, and returns the DOMProcessingInstruction.

Syntax

```
FUNCTION makeProcessingInstruction( n DOMNode)
                                    RETURN DOMProcessingInstruction;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeText()

Description

Casts a given DOMNode to a DOMText, and returns the DOMText.

Syntax

```
FUNCTION makeText(n DOMNode) RETURN DOMText;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

makeDocument()

Description

Casts a given DOMNode to a DOMDocument, and returns the DOMDocument.

Syntax

```
FUNCTION makeDocument(n DOMNode) RETURN DOMDocument;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to cast

writeToFile()

Description

Writes XML node to specified file. The options are given in the following table.

Syntax	Description
PROCEDURE writeToFile(n DOMNode, fileName VARCHAR2);	Writes XML node to specified file using the database character set.
PROCEDURE writeToFile(n DOMNode, fileName VARCHAR2, charset VARCHAR2);	Writes XML node to specified file using the given character set, which is passed in as a separate parameter.

Parameter	IN / OUT	Description
n	(IN)	DOMNode.
fileName	(IN)	File to write to.
charset	(IN)	Given character set.

writeToBuffer()

Description

Writes XML node to specified buffer. The options are given in the following table.

Syntax	Description
PROCEDURE writeToBuffer(n DOMNode, buffer IN OUT VARCHAR2);	Writes XML node to specified buffer using the database character set.
PROCEDURE writeToBuffer(n DOMNode, buffer IN OUT VARCHAR2, charset VARCHAR2);	Writes XML node to specified buffer using the given character set, which is passed in as a separate parameter.

Parameter	IN / OUT	Description
n	(IN)	DOMNode.
buffer	(IN/OUT)	Buffer to write to.
charset	(IN)	Given character set.

writeToClob()

Description
Writes XML node to specified clob. The options are given in the following table.

Syntax	Description
PROCEDURE writeToClob(n DOMNode, cl IN OUT CLOB);	Writes XML node to specified clob using the database character set.
PROCEDURE writeToClob(n DOMNode, cl IN OUT CLOB, charset VARCHAR2);	Writes XML node to specified clob using the given character set, which is passed in as a separate parameter.

Parameter	IN / OUT	Description
n	(IN)	DOMNode.
cl	(IN/OUT)	CLOB to write to.
charset	(IN)	Given character set.

getNodeName()

Description
Get the name of the node depending on its type

Syntax
```
FUNCTION getNodeName(n DOMNode) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getNodeValue()

Description

Get the value of this node, depending on its type.

Syntax

```
FUNCTION getNodeValue(n DOMNode) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
n	IN	DOMNode.

setNodeValue()

Description

Sets the value of this node, depending on its type. When it is defined to be null, setting it has no effect.

Syntax

```
PROCEDURE setNodeValue( n DOMNode,
                        nodeValue IN VARCHAR2);
```

Parameter	IN / OUT	Description
n	IN	DOMNode.
nodeValue	IN	The value to which node is set.

getNodeType()

Description

Retrieves a code representing the type of the underlying object.

Syntax

```
FUNCTION getNodeType(n DOMNode) RETURN NUMBER;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getParentNode()

Description

Retrieves the parent of this node. All nodes, except Attr, Document, DocumentFragment, Entity, and Notation 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

```
FUNCTION getParentNode(n DOMNode) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getChildNodes()

Description

Retrieves a NodeList that contains all children of this node. If there are no children, this is a NodeList containing no nodes.

Syntax

```
FUNCTION getChildNodes(n DOMNode) RETURN DOMNodeList;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getFirstChild()

Description

Retrieves the first child of this node. If there is no such node, this returns null.

Syntax

```
FUNCTION getFirstChild( n DOMNode) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getLastChild()

Description
Retrieves the last child of this node. If there is no such node, this returns `NULL`.

Syntax
```
FUNCTION getLastChild( n DOMNode) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getPreviousSibling()

Description
Retrieves the node immediately preceding this node. If there is no such node, this returns `NULL`.

Syntax
```
FUNCTION getPreviousSibling( n DOMNode) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getNextSibling()

Description
Retrieves the node immediately following this node. If there is no such node, this returns `NULL`.

Syntax
```
FUNCTION getNextSibling( n DOMNode) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getAttributes()

Description

Retrieves a NamedNodeMap containing the attributes of this node (if it is an Element) or null otherwise.

Syntax

```
FUNCTION getAttributes( n DOMNode) RETURN DOMNamedNodeMap;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

getOwnerDocument()

Description

Retrieves the Document object associated with this node. This is also the Document object used to create new nodes. When this node is a Document or a DocumentType which is not used with any Document yet, this is null.

Syntax

```
FUNCTION getOwnerDocument( n DOMNode) RETURN DOMDocument;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

insertBefore()

Description

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.

Syntax

```
FUNCTION insertBefore( n DOMNode,
                       newChild IN DOMNode,
                       refChild IN DOMNode)
                       RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode
newChild	IN	The child to be inserted in the DOMNode `n`
refChild	IN	The reference node before which the newChild is to be inserted

replaceChild()

Description

Replaces the child node `oldChild` with `newChild` in the list of children, and returns the oldChild node.

If `newChild` is a DocumentFragment object, `oldChild` is replaced by all of the DocumentFragment children, which are inserted in the same order. If the newChild is already in the tree, it is first removed.

Syntax

```
FUNCTION replaceChild( n DOMNode,
                       newChild IN DOMNode,
                       oldChild IN DOMNode)
                       RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode
newChild	IN	The new Child which is to replace the oldChild
oldChild	IN	The child of the Node n which is to be replaced

removeChild()

Description

Removes the child node indicated by oldChild from the list of children, and returns it.

Syntax

```
FUNCTION removeChild( n DOMNode,
                      oldChild IN DOMNode)
                      RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode
oldCHild	IN	The child of the node n to be removed

appendChild()

Description

Adds the node newChild to the end of the list of children of this node. If the newChild is already in the tree, it is first removed.

Syntax

```
FUNCTION appendChild( n DOMNode,
                      newChild IN DOMNode)
                      RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode
newChild	IN	The child to be appended to the list of children of Node n

hasChildNodes()

Description

Returns whether this node has any children.

Syntax

```
FUNCTION hasChildNodes( n DOMNode) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
n	IN	DOMNode

cloneNode()

Description

Returns a duplicate of this node; serves as a generic copy constructor for nodes. The duplicate node has no parent; its `parentNode` is `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 an Attribute directly, as opposed to be cloned as part of an Element cloning operation, returns a specified attribute (specified is true). Cloning any other type of node simply returns a copy of this node.

Note that cloning an immutable subtree results in a mutable copy, but the children of an EntityReference clone are read-only. In addition, clones of unspecified Attr nodes are specified. And, cloning Document, DocumentType, Entity, and Notation nodes is implementation dependent.

Syntax

```
FUNCTION cloneNode( n DOMNode,
                    deep boolean)
                    RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	IN	DOMNode
deep	IN	boolean to determine if children are to be cloned or not

DOM Named Node Map Methods

isNull()

Description

Checks that the given DOMNamedNodeMap is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax

```
FUNCTION isNull( nnm DOMNamedNodeMap) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
nnm	(IN)	DOMNameNodeMap to check.

getNamedItem()

Description

Retrieves a node specified by name.

Syntax

```
FUNCTION getNamedItem( nnm DOMNamedNodeMap,
                       name IN VARCHAR2)
                       RETURN DOMNode;
```

Parameter	IN / OUT	Description
nnm	IN	DOMNamedNodeMap
name	IN	Name of the item to be retrieved

setNamedItem()

Description

Adds a node using its nodeName attribute. If a node with that name is already present in this map, it is replaced by the new one.

As the nodeName attribute is used to derive the name under which the node must be stored, multiple nodes of certain types, those that have a “special” string value, cannot be stored because the names would clash. This is seen as preferable to allowing nodes to be aliased.

Syntax

```
FUNCTION setNamedItem( nnm DOMNamedNodeMap,
                       arg IN DOMNode)
                       RETURN DOMNode;
```

Parameter	IN / OUT	Description
nnm	IN	DOMNamedNodeMap
arg	IN	The Node to be added using its NodeName attribute.

removeNamedItem()

Description

Removes a node specified by name. When this map contains the attributes attached to an element, if the removed attribute is known to have a default value, an attribute immediately appears containing the default value as well as the corresponding namespace URI, local name, and prefix when applicable.

Syntax

```
FUNCTION removeNamedItem( nnm DOMNamedNodeMap,
                          name IN VARCHAR2)
                          RETURN DOMNode;
```

Parameter	IN / OUT	Description
nnm	IN	DOMNamedNodeMap
name	IN	The name of the item to be removed from the map

item()

Description

Returns the item in the map which corresponds to the `index` parameter. If index is greater than or equal to the number of nodes in this map, this returns `NULL`.

Syntax

```
FUNCTION item( nnm DOMNamedNodeMap,
               index IN NUMBER)
               RETURN DOMNode;
```

Parameter	IN / OUT	Description
nnm	IN	DOMNamedNodeMap
index	IN	The index in the node map at which the item is to be retrieved

getLength()

Description

The number of nodes in this map. The range of valid child node index is 0 to length-1 inclusive.

Syntax

```
FUNCTION getLength( nnm DOMNamedNodeMap) RETURN NUMBER;
```

Parameter	IN / OUT	Description
nnm	IN	DOMNamedNodeMap

DOM Node List Methods

isNull()

Description

Checks that the given DOMNodeList is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax

```
FUNCTION isNull( nl DOMNodeList) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
nl	(IN)	DOMNodeList to check.

item()

Description

Returns the item in the collection which corresponds to the `index` parameter. If index is greater than or equal to the number of nodes in the list, this returns null.

Syntax

```
FUNCTION item( nl DOMNodeList,
               index IN NUMBER)
               RETURN DOMNode;
```

Parameter	IN / OUT	Description
nl	IN	DOMNodeList
index	IN	The index in the nodelist at which to retrieve the item from

getLength()

Description
The number of nodes in the list. The range of valid child node index is 0 to length-1 inclusive.

Syntax
```
FUNCTION getLength( nl DOMNodeList) RETURN NUMBER;
```

Parameter	IN / OUT	Description
nl	IN	DOMNodeList

DOM Attr Methods

isNull()

Description
Checks that the given DOMAttr is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax
```
FUNCTION isNull( a DOMAttr) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr to check.

makeNode()

Description
Casts given DOMAttr to a DOMNode, and returns the DOMNode.

Syntax
```
FUNCTION makeNode( a DOMAttr) RETURN DOMNode;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr to cast.

getQualifiedName()

Description
Returns the qualified name of the DOMAttr.

Syntax
```
FUNCTION getQualifiedName( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr.

getNamespace()

Description
Returns the namespace of the DOMAttr.

Syntax
```
FUNCTION getNamespace( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr.

getLocalName()

Description
Returns the local name of the DOMAttr.

Syntax
```
FUNCTION getLocalName( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr.

getExpandedName()

Description
Returns the expanded name of the DOMAttr.

Syntax
```
FUNCTION getExpandedName( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	(IN)	DOMAttr.

getName()

Description
Returns the name of this attribute.

Syntax
```
FUNCTION getName( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	IN	DOMAttr

getSpecified()

Description
If this attribute was explicitly given a value in the original document, this is true; otherwise, it is false.

Syntax
```
FUNCTION getSpecified( a DOMAttr) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
a	IN	DOMAttr

getValue()

Description
Retrieves the value of the attribute.

Syntax
```
FUNCTION getValue( a DOMAttr) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
a	IN	DOMAttr

setValue()

Description
Sets the value of the attribute.

Syntax
```
PROCEDURE setValue( a DOMAttr,
                    value IN VARCHAR2);
```

Parameter	IN / OUT	Description
a	IN	DOMAttr
value	IN	The value to set the attribute to

DOM C Data Section Methods

isNull()

Description
Checks that the given DOMCDataSection is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax
```
FUNCTION isNull( cds DOMCDataSection) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
cds	(IN)	DOMCDataSection to check.

makeNode()

Description

Casts the DOMCDataSection to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( cds DOMCDataSection) RETURN DOMNode;
```

Parameter	IN / OUT	Description
cds	(IN)	DOMCDataSection to cast.

Character Data Methods

isNull()

Description

Checks that the given DOMCharacterData is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax

```
FUNCTION isNull( cd DOMCharacterData) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
cd	(IN)	DOMCharacterData to check.

makeNode()

Description

Casts the given DOMCharacterData as a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( cd DOMCharacterData) RETURN DOMNode;
```

Parameter	IN / OUT	Description
cd	(IN)	DOMCharacterData to cast

getData()

Description
Gets the character data of the node that implements this interface.

Syntax
```
FUNCTION getData( cd DOMCharacterData) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData

setData()

Description
Sets the character data of the node that implements this interface.

Syntax
```
PROCEDURE setData( cd DOMCharacterData,
                   data IN VARCHAR2);
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData
data	IN	The data to set the node to

getLength()

Description
The number of 16-bit units that are available through data and the `substringData()` method. This may have the value zero; CharacterData nodes may be empty.

Syntax
```
FUNCTION getLength( cd DOMCharacterData) RETURN NUMBER;
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData

substringData()

Description

Extracts a range of data from the node.

Syntax

```
FUNCTION substringData( cd DOMCharacterData,
                        offset IN NUMBER,
                        cnt IN NUMBER)
                        RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData.
offset	IN	The starting offset of the data from which to get the data.
cnt	IN	The number of characters (from the offset) of the data to get.

appendData()

Description

Appends the string to the end of the character data of the node. Upon success, data provides access to the concatenation of data and the specified string argument.

Syntax

```
PROCEDURE appendData( cd DOMCharacterData,
                      arg IN VARCHAR2);
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData.
arg	IN	The data to append to the existing data.

insertData()

Description

Inserts a string at the specified 16-bit unit offset.

Syntax

```
PROCEDURE insertData( cd DOMCharacterData,
                      offset IN NUMBER,
                        arg IN VARCHAR2);
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData.
offset	IN	The offset at which to insert the data.
arg	IN	The value to be inserted.

deleteData()

Description
Removes a range of 16-bit units from the node. Upon success, data and length reflect the change.

Syntax
```
PROCEDURE deleteData( cd DOMCharacterData,
                      offset IN NUMBER,
                      cnt IN NUMBER);
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData
offset	IN	The offset from which to delete the data
cnt	IN	The number of characters (starting from offset) to delete.

replaceData()

Description
Remove a range of 16-bit units from the node. Upon success, data and length reflect the change.

Syntax
```
PROCEDURE replaceData( cd DOMCharacterData,
                       offset IN NUMBER,
                       cnt IN NUMBER,
                       arg IN VARCHAR2);
```

Parameter	IN / OUT	Description
cd	IN	DOMCharacterData.
offset	IN	The offset at which to replace.
cnt	IN	The no. of characters to replace.
arg	IN	The value to replace with.

DOM Comment Methods

isNull()

Description
Checks that the given DOMComment is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax
```
FUNCTION isNull( com DOMComment) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
com	(IN)	DOMComment to check.

makeNode()

Description
Casts the given DOMComment to a DOMNode, and returns that DOMNode.

Syntax
```
FUNCTION makeNode( com DOMComment) RETURN DOMNode;
```

Parameter	IN / OUT	Description
com	(IN)	DOMComment to cast.

DOM Implementation Methods

isNull()

Description
Checks that the given DOMImplementation is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax
```
FUNCTION isNull( di DOMImplementation) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
di	(IN)	DOMImplementation to check.

hasFeature()

Description
Test if the DOM implementation implements a specific feature.

Syntax
```
FUNCTION hasFeature( di DOMImplementation,
                     feature IN VARCHAR2,
                     version IN VARCHAR2)
                     RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
di	IN	DOMImplementation
feature	IN	The feature to check for
version	IN	The version of the DOM to check in

DOM Document Fragment Methods

isNull()

Description
Checks that the given DOMDocumentFragment is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax

```
FUNCTION isNull( df DOMDocumentFragment) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
df	(IN)	DOMDocumentFragment to check.

makeNode()

Description

Casts the given DOMDocumentFragment to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( df DOMDocumentFragment) RETURN DOMNode;
```

Parameter	IN / OUT	Description
df	(IN)	DOMDocumentFragment to cast.

DOM Document Type Methods

isNull()

Description

Checks that the given DOMDocumentType is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax

```
FUNCTION isNull( dt DOMDocumentType) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
dt	(IN)	DOMDocumentType to check.

makeNode()

Description

Casts the given DOMDocumentType to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( dt DOMDocumentType) RETURN DOMNode;
```

Parameter	IN / OUT	Description
dt	(IN)	DOMDocumentType to cast.

findEntity()

Description

Finds an entity in the given DTD; returns that entity if found.

Syntax

```
FUNCTION findEntity( dt      DOMDocumentType,
                     name    VARCHAR2,
                     par     BOOLEAN)
                     RETURN  DOMEntity;
```

Parameter	IN / OUT	Description
dt	(IN)	The DTD.
name	(IN)	Entity to find.
par	(IN)	Flag to indicate type of entity; TRUE for parameter entity and FALSE for normal entity.

findNotation()

Description

Finds the notation in the given DTD; returns it, if found.

Syntax

```
FUNCTION findNotation( dt   DOMDocumentType,
                       name VARCHAR2)
                       RETURN DOMNotation;
```

Parameter	IN / OUT	Description
dt	(IN)	The DTD.
name	(IN)	The notation to find.

getPublicId()

Description

Returns the public id of the given DTD.

Syntax

```
FUNCTION getPublicId( dt DOMDocumentType) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
dt	(IN)	The DTD.

getSystemId()

Description

Returns the system id of the given DTD.

Syntax

```
FUNCTION getSystemId( dt DOMDocumentType) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
dt	(IN)	The DTD.

writeExternalDTDToFile()

Description

Writes DTD to a specified file. The options are given in the following table.

Syntax	Description
PROCEDURE writeExternalDTDToFile(dt DOMDocumentType, fileName VARCHAR2);	Writes the DTD to a specified file using the database character set.
PROCEDURE writeExternalDTDToFile(dt DOMDocumentType, fileName VARCHAR2, charset VARCHAR2);	Writes the DTD to a specified file using the given character set.

Parameter	IN / OUT	Description
dt	(IN)	The DTD.
fileName	(IN)	The file to write to.
charset	(IN)	Character set.

writeExternalDTDToBuffer()

Description

Writes DTD to a specified buffer. The options are given in the following table.

Syntax	Description
PROCEDURE writeExternalDTDToBuffer(dt DOMDocumentType, buffer IN OUT VARCHAR2);	Writes the DTD to a specified buffer using the database character set.
PROCEDURE writeExternalDTDToBuffer(dt DOMDocumentType, buffer IN OUT VARCHAR2, charset VARCHAR2);	Writes the DTD to a specified buffer using the given character set.

Parameter	IN / OUT	Description
dt	(IN)	The DTD.
buffer	(IN/OUT)	The buffer to write to.
charset	(IN)	Character set.

writeExternalDTDToClob()

Description

Writes DTD to a specified clob. The options are given in the following table.

Syntax	Description
PROCEDURE writeExternalDTDToClob(dt DOMDocumentType, cl IN OUT CLOB);	Writes the DTD to a specified clob using the database character set.

Syntax	Description
PROCEDURE writeExternalDTDToClob(dt DOMDocumentType, cl IN OUT CLOB, charset VARCHAR2);	Writes the DTD to a specified clob using the given character set.

Parameter	IN / OUT	Description
dt	(IN)	The DTD.
cl	(IN/OUT)	The clob to write to.
charset	(IN)	Character set.

getName()

Description

Retrieves the name of DTD, or the name immediately following the DOCTYPE keyword.

Syntax

```
FUNCTION getName( dt DOMDocumentType) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
dt	IN	DOMDocumentType

getEntities()

Description

Retrieves a NamedNodeMap containing the general entities, both external and internal, declared in the DTD.

Syntax

```
FUNCTION getEntities( dt DOMDocumentType) RETURN DOMNamedNodeMap;
```

Parameter	IN / OUT	Description
dt	IN	DOMDocumentType

getNotations()

Description
Retrieves a NamedNodeMap containing the notations declared in the DTD.

Syntax
```
FUNCTION getNotations( dt DOMDocumentType) RETURN DOMNamedNodeMap;
```

Parameter	IN / OUT	Description
dt	IN	DOMDocumentType

DOM Element Methods

isNull()

Description
Checks that the given DOMElement is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax
```
FUNCTION isNull( elem DOMElement) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement to check.

makeNode()

Description
Casts the given DOMElement to a DOMNode, and returns that DOMNode.

Syntax
```
FUNCTION makeNode( elem DOMElement) RETURN DOMNode;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement to cast.

getQualifiedName()

Description
Returns the qualified name of the DOMElement.

Syntax
```
FUNCTION getQualifiedName( elem DOMElement) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement.

getNamespace()

Description
Returns the namespace of the DOMElement.

Syntax
```
FUNCTION getNamespace( elem DOMElement) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement.

getLocalName()

Description
Returns the local name of the DOMElement.

Syntax
```
FUNCTION getLocalName( elem DOMElement) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement.

getExpandedName()

Description
Returns the expanded name of the DOMElement.

Syntax
```
FUNCTION getExpandedName( elem DOMElement) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	DOMElement.

getChildrenByTagName()

Description
Returns the children of the DOMElement. The options are given in the following table.

Syntax	Description
FUNCTION getChildrenByTagName(elem DOMElement, name IN VARCHAR2) RETURN DOMNodeList;	Returns children of the DOMElement given the tag name.
FUNCTION getChildrenByTagName(elem DOMElement, name IN VARCHAR2, ns VARCHAR2) RETURN DOMNodeList;	Returns children of the DOMElement given the tag name and namespace.

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
name	(IN)	Tag name; * matches any tag.
ns	(IN)	Namespace.

getElementsByTagName()

Description

Returns the element children of the DOMElement. The options are given in the following table.

Syntax	Description
FUNCTION getElementsByTagName(elem DOMElement, name IN VARCHAR2) RETURN DOMNodeList;	Returns the element children of the DOMElement given the tag name.
FUNCTION getElementsByTagName(elem DOMElement, name IN VARCHAR2, ns VARCHAR2) RETURN DOMNodeList;	Returns the element children of the DOMElement given the tag name and namespace.

Parameters

elem	(IN)	The DOMElement.
name	(IN)	Tag name; * matches any tag.
ns	(IN)	Namespace.

resolveNamespacePrefix()

Description

Resolves the given namespace prefix, and returns the resolved namespace.

Syntax

```
FUNCTION resolveNamespacePrefix( elem DOMElement,
                                 prefix VARCHAR2)
                                 RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
prefix	(IN)	Namespace prefix.

getTagName()

Description
Returns the name of the DOMElement.

Syntax
```
FUNCTION getTagName(elem DOMElement) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.

getAttribute()

Description
Returns the value of a DOMElement's attribute by name.

Syntax
```
FUNCTION getAttribute( elem DOMElement,
                       name IN VARCHAR2)
                       RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
name	(IN)	Attribute name; * matches any attribute.

setAttribute()

Description
Sets the value of a DOMElement's attribute by name.

Syntax
```
PROCEDURE setAttribute( elem DOMElement,
                        name IN VARCHAR2,
                        value IN VARCHAR2);
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.

Parameter	IN / OUT	Description
name	(IN)	Attribute name; * matches any attribute.
value	(IN)	Attribute value

removeAttribute()

Description

Removes an attribute from the DOMElement by name.

Syntax

```
PROCEDURE removeAttribute( elem DOMElement,
                           name IN VARCHAR2);
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
name	(IN)	Attribute name; * matches any attribute.

getAttributeNode()

Description

Returns an attribute node from the DOMElement by name.

Syntax

```
FUNCTION getAttributeNode( elem DOMElement,
                           name IN VARCHAR2)
                           RETURN DOMAttr;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
name	(IN)	Attribute name; * matches any attribute.

setAttributeNode()

Description

Adds a new attribute node to the DOMElement.

Syntax

```
FUNCTION setAttributeNode( elem DOMElement,
                           newAttr IN DOMAttr)
                           RETURN DOMAttr;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
newAttr	(IN)	The new DOMAttr.

removeAttributeNode()

Description

Removes the specified attribute node from the DOMElement.

Syntax

```
FUNCTION removeAttributeNode( elem DOMElement,
                              oldAttr IN DOMAttr)
                              RETURN DOMAttr;
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.
oldAttr	(IN)	The old DOMAttr.

normalize()

Description

Normalizes the text children of the DOMElement.

Syntax

```
PROCEDURE normalize( elem DOMElement);
```

Parameter	IN / OUT	Description
elem	(IN)	The DOMElement.

DOM Entity Methods

isNull()

Description

Checks that the given DOMEntity is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( ent DOMEntity) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
ent	(IN)	DOMEntity to check.

makeNode()

Description

Casts given DOMEntity to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( ent DOMEntity) RETURN DOMNode;
```

Parameter	IN / OUT	Description
ent	(IN)	DOMEntity to cast.

getPublicId()

Description

Returns the public identifier of the DOMEntity.

Syntax

```
FUNCTION getPublicId( ent DOMEntity) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
ent	(IN)	DOMEntity.

getSystemId()

Description

Returns the system identifier of the DOMEntity.

Syntax

```
FUNCTION getSystemId( ent DOMEntity) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
ent	(IN)	DOMEntity.

getNotationName()

Description

Returns the notation name of the DOMEntity.

Syntax

```
FUNCTION getNotationName( ent DOMEntity) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
ent	(IN)	DOMEntity.

DOM Entity Reference Methods

isNull()

Description

Checks that the given DOMEntityRef is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax

```
FUNCTION isNull( eref DOMEntityReference) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
eref	(IN)	DOMEntityReference to check.

makeNode()

Description
Casts the DOMEntityReference to a DOMNode, and returns that DOMNode.

Syntax
```
FUNCTION makeNode( eref DOMEntityReference) RETURN DOMNode;
```

Parameter	IN / OUT	Description
eref	(IN)	DOMEntityReference to cast.

DOM Notation Methods

isNull()

Description
Checks that the given DOMNotation is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax
```
FUNCTION isNull( n DOMNotation) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNotation to check.

makeNode()

Description
Casts the DOMNotation to a DOMNode, and returns that DOMNode.

Syntax
```
FUNCTION makeNode( n DOMNotation) RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNotation to cast.

getPublicId()

Description

Returns the public identifier of the DOMNotation.

Syntax

```
FUNCTION getPublicId( n DOMNotation) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNotation.

getSystemId()

Description

Returns the system identifier of the DOMNotation.

Syntax

```
FUNCTION getSystemId( n DOMNotation) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNotation.

DOM Processing Instruction Methods

isNull()

Description

Checks that the given DOMProcessingInstruction is `NULL`; returns `TRUE` if it is `NULL`, `FALSE` otherwise.

Syntax

```
FUNCTION isNull( pi DOMProcessingInstruction) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
pi	(IN)	DOMProcessingInstruction to check.

makeNode()

Description
Casts the DOMProcessingInstruction to a DOMNode, and returns that DOMNode.

Syntax
```
FUNCTION makeNode( pi DOMProcessingInstruction) RETURN DOMNode;
```

Parameter	IN / OUT	Description
pi	(IN)	DOMProcessingInstruction to cast.

getData()

Description
Returns the content data of the DOMProcessingInstruction.

Syntax
```
FUNCTION  getData( pi DOMProcessingInstruction) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
pi	(IN)	DOMProcessingInstruction.

getTarget()

Description
Returns the target of the DOMProcessingInstruction.

Syntax
```
FUNCTION getTarget( pi DOMProcessingInstruction) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
pi	(IN)	DOMProcessingInstruction.

setData()

Description
Sets the content data of the DOMProcessingInstruction.

Syntax

```
PROCEDURE setData( pi DOMProcessingInstruction,
                   data IN VARCHAR2);
```

Parameter	IN / OUT	Description
pi	(IN)	DOMProcessingInstruction.
data	(IN)	New processing instruction content data.

DOM Text Methods

isNull()

Description

Checks that the given DOMText is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( t DOMText) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
t	(IN)	DOMText to check.

makeNode()

Description

Casts the DOMText to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( t DOMText) RETURN DOMNode;
```

Parameter	IN / OUT	Description
t	(IN)	DOMText to cast.

splitText()

Description

Breaks this DOMText node into two DOMText nodes at the specified offset.

Syntax

```
FUNCTION splitText( t DOMText,
                    offset IN NUMBER)
                    RETURN DOMText;
```

Parameter	IN / OUT	Description
t	(IN)	DOMText
offset	(IN)	Offset at which to split.

DOM Document Methods

isNull()

Description

Checks that the given DOMDocument is NULL; returns TRUE if it is NULL, FALSE otherwise.

Syntax

```
FUNCTION isNull( doc DOMDocument) RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument to check.

makeNode()

Description

Casts the DOMDocument to a DOMNode, and returns that DOMNode.

Syntax

```
FUNCTION makeNode( doc DOMDocument) RETURN DOMNode;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument to cast.

newDOMDocument()

Description
Returns a new DOMDocument instance.

Syntax
```
FUNCTION newDOMDocument RETURN DOMDocument;
```

freeDocument()

Description
Frees DOMDocument object.

Syntax
```
PROCEDURE freeDocument( doc DOMDocument);
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

getVersion()

Description
Returns the version information for the XML document.

Syntax
```
FUNCTION getVersion( doc DOMDocument) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

setVersion()

Description
Sets version information for the XML document.

Syntax

```
PROCEDURE setVersion( doc     DOMDocument,
                      version VARCHAR2);
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
version	((N)	Version information.

getCharset()

Description

Retrieves the character set of the XML document.

Syntax

```
FUNCTION getCharset( doc DOMDocument) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

setCharset()

Description

Sets character set of the XML document.

Syntax

```
PROCEDURE setCharset( doc DOMDocument,
                      charset VARCHAR2);
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
charset	((N)	Character set.

getStandalone()

Description

Retrieves standalone information for the XML document.

Syntax

```
FUNCTION getStandalone( doc DOMDocument) RETURN VARCHAR2;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

setStandalone()

Description

Sets standalone information for the XML document.

Syntax

```
PROCEDURE setStandalone( doc DOMDocument,
                         value VARCHAR2);
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
value	((N)	Standalone information

writeToFile()

Description

Writes XML document to a specified file. The options are given in the following table.

Syntax	Description
PROCEDURE writeToFile(doc DOMDocument, fileName VARCHAR2);	Writes XML document to a specified file using database character set.
PROCEDURE writeToFile(doc DOMDocument, fileName VARCHAR2, charset VARCHAR2);	Writes XML document to a specified file using given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
filename	(N)	File to write to.
charset	(IN)	Character set.

writeToBuffer()

Description

Writes XML document to a specified buffer. The options are given in the following table.

Syntax	Description
PROCEDURE writeToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2);	Writes XML document to a specified buffer using database character set.
PROCEDURE writeToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2, charset VARCHAR2);	Writes XML document to a specified buffer using given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
buffer	(N/OUT)	Buffer to write to.
charset	(IN)	Character set.

writeToClob()

Description

Writes XML document to a specified clob. The options are given in the following table.

Syntax	Description
PROCEDURE writeToClob(doc DOMDocument, cl IN OUT CLOB);	Writes XML document to a specified clob using database character set.
PROCEDURE writeToClob(doc DOMDocument, cl IN OUT CLOB, charset VARCHAR2);	Writes XML document to a specified clob using given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
cl	(N/OUT)	Buffer to write to.
charset	(IN)	Character set.

writeExternalDTDToFile()

Description

Writes an external DTD to specified file. The options are given in the following table.

Syntax	Description
PROCEDURE writeExternalDTDToFile(doc DOMDocument, fileName VARCHAR2);	Writes an external DTD to specified file using the database character set.
PROCEDURE writeExternalDTDToFile(doc DOMDocument, fileName VARCHAR2, charset VARCHAR2);	Writes an external DTD to specified file using the given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
fileName	(N)	File to write to.

Parameter	IN / OUT	Description
charset	(IN)	Character set.

writeExternalDTDToBuffer()

Description

Writes an external DTD to specified buffer. The options are given in the following table.

Syntax	Description
PROCEDURE writeExternalDTDToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2);	Writes an external DTD to specified buffer using the database character set.
PROCEDURE writeExternalDTDToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2, charset VARCHAR2);	Writes an external DTD to specified buffer using the given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
buffer	(N/OUT)	Buffer to write to.
charset	(IN)	Character set.

writeExternalDTDToClob()

Description

Writes an external DTD to specified clob. The options are given in the following table.

Syntax	Description
PROCEDURE writeExternalDTDToClob(doc DOMDocument, cl IN OUT CLOB);	Writes an external DTD to specified clob using the database character set.

Syntax	Description
PROCEDURE writeExternalDTDToClob(doc DOMDocument, cl IN OUT CLOB, charset VARCHAR2);	Writes an external DTD to specified clob using the given character set.

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
cl	(N)	Clob to write to.
charset	(IN)	Character set.

getDoctype()

Description
Returns the DTD associated to the DOMDocument.

Syntax
```
FUNCTION getDoctype( doc DOMDocument) RETURN DOMDocumentType;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

getImplementation()

Description
Returns the DOMImplementation object that handles this DOMDocument.

Syntax
```
FUNCTION getImplementation( doc DOMDocument) RETURN DOMImplementation;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

getDocumentElement()

Description

Returns the child node, or the document element of the DOMDocument.

Syntax

```
FUNCTION getDocumentElement( doc DOMDocument) RETURN DOMElement;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

createElement()

Description

Creates a DOMElement.

Syntax

```
FUNCTION createElement( doc DOMDocument,
                        tagName IN VARCHAR2)
                        RETURN DOMElement;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
tagName	(IN)	Tagname for new DOMElement.

createDocumentFragment()

Description

Creates a DOMDocumentFragment.

Syntax

```
FUNCTION createDocumentFragment( doc DOMDocument) RETURN DOMDocumentFragment;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.

createTextNode()

Description
Creates a DOMText node.

Syntax
```
FUNCTION createTextNode( doc DOMDocument,
                         data IN VARCHAR2)
                         RETURN DOMText;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
data	(IN)	Content of the DOMText node.

createComment()

Description
Creates a DOMComment node.

Syntax
```
FUNCTION createComment( doc DOMDocument,
                        data IN VARCHAR2)
                        RETURN DOMComment;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
data	(IN)	Content of the DOMComment node.

createCDATASection()

Description
Creates a DOMCDATASection node.

Syntax
```
FUNCTION createCDATASection( doc DOMDocument,
                             data IN VARCHAR2)
                             RETURN DOMCDATASection;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
data	(IN)	Content of the DOMCDATASection node.

createProcessingInstruction()

Description

Creates a DOMProcessingInstruction node.

Syntax

```
FUNCTION createProcessingInstruction( doc DOMDocument,
                                      target IN VARCHAR2,
                                      data IN VARCHAR2)
                                      RETURN DOMProcessingInstruction;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
target	(IN)	Target of the new processing instruction.
data	(IN)	Content data of the new processing instruction.

createAttribute()

Description

Creates a DOMAttr node.

Syntax

```
FUNCTION createAttribute( doc DOMDocument,
                          name IN VARCHAR2)
                          RETURN DOMAttr;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
name	(IN)	New attribute name.

createEntityReference()

Description

Creates a DOMEntityReference node.

Syntax

```
FUNCTION createEntityReference( doc DOMDocument,
                                name IN VARCHAR2)
                                RETURN DOMEntityReference;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
name	(IN)	New entity reference name.

getElementsByTagName()

Description

Returns a DOMNodeList of all the elements with a given tagname.

Syntax

```
FUNCTION getElementsByTagName( doc DOMDocument,
                               tagname IN VARCHAR2)
                               RETURN DOMNodeList;
```

Parameter	IN / OUT	Description
doc	(IN)	DOMDocument.
tagname	(IN)	Name of the tag to match on.

26

PL/SQL Parser API for XMLType

The user can access the contents and structure of XML documents through the APIs of the DBMS_XMLPARSER Package.

This chapter discusses the following topics:

- Description of DBMS_XMLPARSER
- Subprograms of DBMS_XMLPARSER

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

DBMS_XMLPARSER Package

Description of DBMS_XMLPARSER

The Extensible Markup Language (XML) describes a class of data objects called XML documents. It partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of the Standard Generalized Markup Language (SGML). 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 PL/SQL 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 for this PL/SQL XML parser:

- A parse tree which can be accessed by DOM APIs is built
- The parser is validating if a DTD is found, otherwise, it is non-validating
- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

Subprograms of DBMS_XMLPARSER

Table 26–1 Summary of Subprograms of DBMS_XMLPARSER

Subprogram	Description
parse() on page 26-3	Parses XML stored in the given url/file.
newParser() on page 26-4	Returns a new parser instance
parseBuffer() on page 26-4	Parses XML stored in the given buffer
parseClob() on page 26-4	Parses XML stored in the given clob

Table 26–1 Summary of Subprograms of DBMS_XMLPARSER (Cont.)

Subprogram	Description
parseDTD() on page 26-5	Parses DTD stored in the given url/file
parseDTDBuffer() on page 26-5	Parses DTD stored in the given buffer
parseDTDClob() on page 26-6	Parses DTD stored in the given clob
setBaseDir() on page 26-6	Sets base directory used to resolve relative URLs.
showWarnings() on page 26-7	Turns warnings on or off.
setErrorLog() on page 26-7	Sets errors to be sent to the specified file
setPreserveWhitespace() on page 26-8	Sets white space preserve mode
setValidationMode() on page 26-8	Sets validation mode.
getValidationMode() on page 26-8	Returns validation mode.
setDoctype() on page 26-9	Sets DTD.
getDoctype() on page 26-9	Gets DTD Parser.
getDocument() on page 26-10	Gets DOM document.
freeParser() on page 26-10	Frees a parser object.
getReleaseVersion() on page 26-10	Returns the release version of Oracle XML Parser for PL/SQL.

parse()

Description

Parses XML stored in the given url/file. An application error is raised if parsing fails. The options are described in the following table.

Syntax	Description
FUNCTION parse(url VARCHAR2) RETURN DOMDocument;	Returns the built DOM Document. This is meant to be used when the default parser behavior is acceptable and just a url/file needs to be parsed.
PROCEDURE parse(p Parser, url VARCHAR2);	Any changes to the default parser behavior should be effected before calling this procedure.

Parameter	IN / OUT	Description
url	(IN)	Complete path of the url/file to be parsed.
p	(IN)	Parser instance.

newParser()

Description

Returns a new parser instance. This function must be called before the default behavior of Parser can be changed and if other parse methods need to be used.

Syntax

```
FUNCTION newParser RETURN Parser;
```

parseBuffer()

Description

Parses XML stored in the given buffer. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE parseBuffer( p   Parser,
                       doc VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
doc	(IN)	XML document buffer to parse.

parseClob()

Description

Parses XML stored in the given clob. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE parseClob( p   Parser,
                     doc CLOB);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
doc	(IN)	XML document buffer to parse.

parseDTD()

Description

Parses the DTD stored in the given url/file. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE parseDTD( p     Parser,
                    url   VARCHAR2,
                    root  VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
url	(IN)	Complete path of the url/file to be parsed.
p	(IN)	Parser instance.

parseDTDBuffer()

Description

Parses the DTD stored in the given buffer. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE parseDTDBuffer( p    Parser,
                          dtd  VARCHAR2,
                          root VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD buffer to parse.
root	(IN)	Name of the root element.

parseDTDClob()

Description

Parses the DTD stored in the given clob. Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing fails.

Syntax

```
PROCEDURE parseDTDClob( p    Parser,
                        dtd  CLOB,
                        root VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD Clob to parse.
root	(IN)	Name of the root element.

setBaseDir()

Description

Sets base directory used to resolve relative URLs. An application error is raised if parsing fails.

Syntax

```
PROCEDURE setBaseDir( p   Parser,
                      dir VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dir	(IN)	Directory used as a base directory.

showWarnings()

Description

Turns warnings on or off.

Syntax

```
PROCEDURE showWarnings( p   Parser,
                        yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - show warnings, FALSE - don't show warnings.

setErrorLog()

Description

Sets errors to be sent to the specified file

Syntax

```
PROCEDURE setErrorLog( p        Parser,
                       fileName VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
fileName	(IN)	Complete path of the file to use as the error log.

setPreserveWhitespace()

Description

Sets whitespace preserving mode.

Syntax

```
PROCEDURE setPreserveWhitespace( p   Parser,
                                 yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - preserve, FALSE - don't preserve.

setValidationMode()

Description

Sets validation mode.

Syntax

```
PROCEDURE setValidationMode( p   Parser,
                             yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
yes	(IN)	Mode to set: TRUE - validate, FALSE - don't validate.

getValidationMode()

Description

Retrieves validation mode; TRUE for validating, FALSE otherwise.

Syntax

```
FUNCTION getValidationMode( p Parser)
                          RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

setDoctype()

Description

Sets a DTD to be used by the parser for validation. This call should be made before the document is parsed.

Syntax

```
PROCEDURE setDoctype( p   Parser,
                      dtd DOMDocumentType);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.
dtd	(IN)	DTD to set.

getDoctype()

Description

Returns the parsed DTD; this function MUST be called only after a DTD is parsed.

Syntax

```
FUNCTION getDoctype( p Parser)
                   RETURN DOMDocumentType;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

getDocument()

Description

Returns the root of the DOM tree document built by the parser; this function MUST be called only after a document is parsed.

Syntax

```
FUNCTION getDocument( p Parser)
                    RETURN DOMDocument;
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

freeParser()

Description

Frees a parser object.

Syntax

```
PROCEDURE freeParser( p Parser);
```

Parameter	IN / OUT	Description
p	(IN)	Parser instance.

getReleaseVersion()

Returns the release version of the Oracle XML parser for PL/SQL.

Syntax

```
PROCEDURE getReleaseVersion RETURN VARCHAR2;
```

27

PL/SQL XSLT Processing for XMLType

The Extensible Stylesheet Language (XSL) Package Processor APIs in the DBMS_XSLPROCESSOR Package enables PL/SQL developers to access the contents and structure of XML documents.

This chapter discusses the following topics:

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

DBMS_XSLPROCESSOR Package

Description of DBMS_XSLPROCESSOR

The Extensible Stylesheet Language Transformation (XSLT), describes rules for transforming a source tree into a result tree. A transformation expressed in XSLT is called a stylesheet. The transformation specified is achieved by associating patterns with templates defined in the stylesheet. A template is instantiated to create part of the result tree. This PL/SQL implementation of the XSL processor followed the W3C XSLT working draft (rev WD-xslt-19990813) and included the required behavior of an XSL processor in terms of how it must read XSLT stylesheets and the transformation it must effect.

The following is the default behavior for this PL/SQL XSL Processor:

- A result tree which can be accessed by DOM APIs is built
- Errors are not recorded unless an error log is specified; however, an application error will be raised if parsing fails

Subprograms of DBMS_XSLPROCESSOR

Table 27–1 Summary of Subprograms of DBMS_XSLPROCESSOR

Subprogram	Description
newProcessor() on page 27-3	Returns a new processor instance.
processXSL() on page 27-3	Transforms an input XML document.
showWarnings() on page 27-6	Turns warnings on or off.
setErrorLog() on page 27-6	Sets errors to be sent to the specified file.
newStylesheet() on page 27-6	Creates a new stylesheet using the given input and reference URLs.
transformNode() on page 27-7	Transforms a node in a DOM tree using the given stylesheet.
selectNodes() on page 27-8	Selects nodes from a DOM tree that match the given pattern.
selectSingleNodes() on page 27-8	Selects the first node from the tree that matches the given pattern.
valueOf() on page 27-8	Retrieves the value of the first node from the tree that matches the given pattern

Table 27–1 Summary of Subprograms of DBMS_XSLPROCESSOR (Cont.)

Subprogram	Description
setParam() on page 27-9	Sets a top-level parameter in the stylesheet
removeParam() on page 27-9	Removes a top-level stylesheet parameter
resetParams() on page 27-10	Resets the top-level stylesheet parameters
freeStylesheet() on page 27-10	Frees a stylesheet object
freeProcessor() on page 27-10	Frees a processor object

newProcessor()

Description

Returns a new processor instance. This function must be called before the default behavior of Processor can be changed and if other processor methods need to be used.

Syntax

```
FUNCTION newProcessor RETURN Processor;
```

processXSL()

Description

Transforms input XML document. Any changes to the default processor behavior should be effected before calling this procedure. An application error is raised if processing fails. The options are described in the following table.

Syntax	Description
FUNCTION processXSL(p Processor, ss Stylesheet, xmldoc DOMDocument), RETURN DOMDocumentFragment;	Transforms input XML document using given DOMDocument and stylesheet, and returns the resultant document fragment.

Syntax	Description
FUNCTION processXSL(p Processor, ss Stylesheet, url VARCHAR2, RETURN DOMDocumentFragment;	Transforms input XML document using given document as URL and the stylesheet, and returns the resultant document fragment.
FUNCTION processXSL(p Processor, ss Stylesheet, clb CLOB) RETURN DOMDocumentFragment;	Transforms input XML document using given document as CLOB and the stylesheet, and returns the resultant document fragment.
PROCEDURE processXSL(p Processor, ss Stylesheet, xmldoc DOMDocument, dir VARCHAR2, fileName VARCHAR2);	Transforms input XML document using given DOMDocument and the stylesheet, and writes the output to the specified file.
PROCEDURE processXSL(p Processor, ss Stylesheet, url VARCHAR2, dir VARCHAR2, fileName VARCHAR2);	Transforms input XML document using given URL and the stylesheet, and writes the output to the specified file in a specified directory.
PROCEDURE processXSL(p Processor, ss Stylesheet, xmldoc DOMDocument, cl IN OUT CLOB);	Transforms input XML document using given DOMDocument and the stylesheet, and writes the output to a CLOB.
FUNCTION processXSL(p Processor, ss Stylesheet, xmldf DOMDocumentFragment) RETURN DOMDocumentFragment;	Transforms input XML DocumentFragment using given DOMDocumentFragment and the stylesheet, and returns the resultant document fragment.

Syntax	Description
PROCEDURE processXSL(p Processor, ss Stylesheet, xmldf DOMDocumentFragment, dir VARCHAR2, fileName VARCHAR2);	Transforms input XML DocumentFragment using given DOMDocumentFragment and the stylesheet, and writes the output to the specified file in a specified directory.
PROCEDURE processXSL(p Processor, ss Stylesheet, xmldf DOMDocumentFragment, buf IN OUT VARCHAR2);	Transforms input XML DocumentFragment using given DOMDocumentFragment and the stylesheet, and writes the output to a buffer.
PROCEDURE processXSL(p Processor, ss Stylesheet, xmldf DOMDocumentFragment, cl IN OUT CLOB);	Transforms input XML DocumentFragment using given DOMDocumentFragment and the stylesheet, and writes the output to a CLOB.

Parameter	IN / OUT	Description
p	(IN)	Processor instance.
ss	(IN)	Stylesheet instance.
xmldoc	(IN)	XML document being transformed.
url	(IN)	URL for the information being transformed.
clb	(IN)	CLOB containing information to be transformed.
dir	(IN)	Directory where processing output file is saved.
fileName	(IN)	Processing output file.
cl	(IN/OUT)	CLOB to which the processing output is saved.
buf	(IN/OUT)	Buffer to which the processing output is saved.
xmldf	(IN)	XML document fragment being transformed.

showWarnings()

Description

Turns warnings on (TRUE) or off (FALSE).

Syntax

```
PROCEDURE showWarnings( p Processor,
                        yes BOOLEAN);
```

Parameter	IN / OUT	Description
p	(IN)	Processor instance.
yes	(IN)	Mode to set: TRUE to show warnings, FALSE otherwise

setErrorLog()

Description

Sets errors to be sent to the specified file.

Syntax

```
PROCEDURE setErrorLog( p Processor,
                       fileName VARCHAR2);
```

Parameter	IN / OUT	Description
p	(IN)	Processor instance.
fileName	(IN)	complete path of the file to use as the error log.

newStylesheet()

Description

Creates and returns a new stylesheet instance. The options are described in the following table.

Syntax	Description
FUNCTION newStylesheet(xmldoc DOMDocument, ref VARCHAR2) RETURN Stylesheet;	Creates and returns a new stylesheet instance using the given DOMDocument and reference URLs.
FUNCTION newStylesheet(inp VARCHAR2, ref VARCHAR2) RETURN Stylesheet;	Creates and returns a new stylesheet instance using the given input and reference URLs.

Parameter	IN / OUT	Description
xmldoc	(IN)	DOMDocument to use for construction.
inp	(IN)	Input URL to use for construction.
ref	(IN)	Reference URL

transformNode()

Description

Transforms a node in a DOM tree using the given stylesheet, and returns the result of the transformation as a DOMDocumentFragment.

Syntax

```
FUNCTION transformNode( n DOMNode,
                        ss Stylesheet)
                        RETURN DOMDocumentFragment;
```

Parameter	IN / OUT	Description
n	(IN)	DOMNode to transform.
ss	(IN)	Stylesheet to use.

selectNodes()

Description

Selects nodes which match the given pattern from a DOM tree, and returns the result of the selection.

Syntax

```
FUNCTION selectNodes( n DOMNode,
                      pattern VARCHAR2)
                      RETURN DOMNodeList;
```

Parameter	IN / OUT	Description
n	(IN)	Root DOMNode of the tree.
pattern	(IN)	Pattern to use.

selectSingleNodes()

Description

Selects the first node from the tree that matches the given pattern, and returns that node.

Syntax

```
FUNCTION selectSingleNodes( n DOMNode,
                            pattern VARCHAR2)
                            RETURN DOMNode;
```

Parameter	IN / OUT	Description
n	(IN)	Root DOMNode of the tree.
pattern	(IN)	Pattern to use.

valueOf()

Description

Retrieves the value of the first node from the tree that matches the given pattern.

Syntax

```
PROCEDURE valueOf( n DOMNode,
                   pattern VARCHAR2,
                   val OUT VARCHAR2);
```

Parameter	IN / OUT	Description
n	(IN)	Root DOMNode of the tree.
pattern	(IN)	Pattern to use.
val	(OUT)	Retrieved value.

setParam()

Description

Sets a top level parameter in the stylesheet. The parameter value must be a valid XPath expression. Literal string values must be quoted.

Syntax

```
PROCEDURE setParam( ss Stylesheet,
                    name VARCHAR2,
                    value VARCHAR2);
```

Parameter	IN / OUT	Description
ss	(IN)	Stylesheet.
name	(IN)	Name of the parameter.
value	(IN)	Value of the parameter.

removeParam()

Description

Removes a top level stylesheet parameter.

Syntax

```
PROCEDURE removeParam( ss Stylesheet,
                       name VARCHAR2);
```

Parameter	IN / OUT	Description
ss	(IN)	Stylesheet.
name	(IN)	Name of the parameter.

resetParams()

Description

Resets the top-level stylesheet parameters.

Syntax

```
PROCEDURE resetParams( ss Stylesheet);
```

Parameter	IN / OUT	Description
ss	(IN)	Stylesheet.

freeStylesheet()

Description

Frees a Stylesheet object.

Syntax

```
PROCEDURE freestylesheet( ss Stylesheet);
```

Parameter	IN / OUT	Description
ss	(IN)	Stylesheet.

freeProcessor()

Description

Frees a Processor object.

Syntax

```
PROCEDURE freeProccessor( p Processor);
```

Parameter	IN / OUT	Description
p	(IN)	Processor.

28

DBMS_XMLSCHEMA and Catalog Views for PL/SQL

This chapter contains the following sections:

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

DBMS_XMLSCHEMA Package

Description of DBMS_XMLSCHEMA

This package is created by script dbmsxsch.sql during Oracle XML DB installation. It provides procedures to register and delete XML schemas.

Constants of DBMS_XMLSCHEMA

Table 28–1 Constants of DBMS_XMLSCHEMA

Constant	Description
DELETE_RESTRICT	CONSTANT NUMBER := 1;
DELETE_INVALIDATE	CONSTANT NUMBER := 2;
DELETE_CASCADE	CONSTANT NUMBER := 3;
DELETE_CASCADE_FORCE	CONSTANT NUMBER := 4;

Procedures and Functions of DBMS_XMLSCHEMA

Table 28–2 Summary of Functions and Procedures of DBMS_XMLSCHEMA

Constant	Description
registerSchema() on page 28-3	Registers the specified schema for use by Oracle. This schema can then be used to store documents conforming to this.
registerURI() on page 28-5	Registers an XMLSchema specified by a URI name.
deleteSchema() on page 28-6	Removes the schema from Oracle XML DB.
generateBean() on page 28-7	Generates the Java bean code corresponding to a registered XML schema
compileSchema() on page 28-8	Used to re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state.
generateSchema() on page 28-8	Generates XML schema(s) from an oracle type name.

registerSchema()

Description

Registers the specified schema for use by the Oracle XML DB. The available options are given in the following table.

Syntax	Description
procedure registerSchema(schemaURL IN varchar2, schemaDoc IN VARCHAR2, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, genTables IN BOOLEAN := TRUE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null);	Registers a schema specified as a VARCHAR2.
procedure registerSchema(schemaURL IN varchar2, schemaDoc IN CLOB, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FASLE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null);	Registers the schema specified as a CLOB.
procedure registerSchema(schemaURL IN varchar2, schemaDoc IN BFILE, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null);	Registers the schema specified as a BFILE.

Syntax	Description
procedure registerSchema(schemaURL IN varchar2, schemaDoc IN SYS.XMLType, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null);	Registers the schema specified as an XMLType.
procedure registerSchema(schemaURL IN varchar2, schemaDoc IN SYS.URIType, local IN BOOLEAN := TRUE, genTypes IN BOOLEAN := TRUE, genbean IN BOOLEAN := FALSE, force IN BOOLEAN := FALSE, owner IN VARCHAR2 := null);	Registers the schema specified as a URIType.

Parameter	IN / OUT	Description
schemaURL	(IN)	URL that uniquely identifies the schema document. This value is used to derive the path name of the schema document within the Oracle XML DB hierarchy.
schemaDoc	(IN)	a valid XML schema document
local	(IN)	Is this a local or global schema? By default, all schemas are registered as local schemas, under `/sys/schemas/<username/...` If a schema is registered as global, it is added under `/sys/schemas/PUBLIC/....` Write privileges on `PUBLIC` directory are necessary to register a schema as global.
genTypes	(IN)	Should the schema compiler generate object types? By default, `TRUE`.
genbean	(IN)	Should the schema compiler generate Java beans? By default, `FALSE`.
genTables	(IN)	Should the schema compiler generate default tables? By default, `TRUE`.

Parameter	IN / OUT	Description
force	(IN)	If this parameter is set to TRUE, the schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors. By default, the value of this parameter is FALSE.
owner	(IN)	This parameter specifies the name of the database user owning the XML schema object. By default, the user registering the schema owns the XML schema object. This parameter can be used to register a XML schema to be owned by a different database user.

registerURI()

Description

Registers an XMLSchema specified by a URI name.

Syntax

```
procedure registerURI(schemaURL IN varchar2,
                      schemaDocURI IN varchar2,
                      local IN BOOLEAN := TRUE,
                      genTypes IN BOOLEAN := TRUE,
                      genbean IN BOOLEAN := FALSE,
                      genTables IN BOOLEAN := TRUE,
                      force IN BOOLEAN := FALSE,
                      owner IN VARCHAR2 := null);
```

Parameter	IN / OUT	Description
schemaURL	(IN)	A name that uniquely identifies the schema document.
schemaDocURI	(IN)	Pathname (URI) corresponding to the physical location of the schema document. The URI path could be based on HTTP, FTP, DB or Oracle XML DB protocols. This function constructs a URIType instance using the URIFactory - and invokes the `registerSchema()` function.

Parameter	IN / OUT	Description
local	(IN)	Is this a local or global schema? By default, all schemas are registered as local schemas under `/sys/schemas/<username/...` If a schema is regsitered as global, it is added under `/sys/schemas/PUBLIC/...` User needs write privileges on the PUBLIC directory to be able to register a schema as global.
genTypes	(IN)	Should the schema compiler generate object types? By default, `TRUE`.
genbean	(IN)	Should the schema compiler generate Java beans? By default, `FALSE`.
genTables	(IN)	Should the schema compiler generate default tables? By default, `TRUE`.
force	(IN)	If this parameter is set to TRUE, the schema registration will not raise errors. Instead, it creates an invalid XML schema object in case of any errors. By default, the value of this parameter is `FALSE`.
owner	(IN)	This parameter specifies the name of the database user owning the XML schema object. By default, the user registering the schema owns the XML schema object. This parameter can be used to register a XML schema to be owned by a different database user.

deleteSchema()

Description

Deletes the XMLSchema specified by the URL. Can result in a `ORA-31001` exception: invalid resource handle or path name.

Syntax

```
procedure deleteSchema( schemaURL IN varchar2,
                        delete_option IN pls_integer := DELETE_RESTRICT);
```

Parameter	IN / OUT	Description
schemaURL	(IN)	URL identifying the schema to be deleted.

Parameter	IN / OUT	Description
delete_option	(IN)	Option for deleting schema.

Options for delete_option parameter

Option	Description
DELETE_RESTRICT	Schema deletion fails if there are any tables or schemas that depend on this schema.
DELETE_INVALIDATE	Schema deletion does not fail if there are any dependencies. Instead, it simply invalidates all dependent objects.
DELETE_CASCADE	Schema deletion will also drop all default SQL types and default tables. However the deletion fails if there are any stored instances conforming to this schema.
DELETE_CASCADE_FORCE	Similar to CASCADE except that it does not check for any stored instances conforming to this schema. Also it ignores any errors.

generateBean()

Description

This procedure can be used to generate the Java bean code corresponding to a registered XML schema. Note that there is also an option to generate the beans as part of the registration procedure itself. Can result in a `ORA-31001` exception: invalid resource handle or path name.

Syntax

```
procedure generateBean(schemaURL IN varchar2);
```

Parameter	IN / OUT	Description
schemaURL	(IN)	Name identifying a registered XML schema.

compileSchema()

Description

This procedure can be used to re-compile an already registered XML schema. This is useful for bringing a schema in an invalid state to a valid state. Can result in a `ORA-31001` exception: invalid resource handle or path name.

Syntax

```
procedure compileSchema( schemaURL IN varchar2);
```

Parameter	IN / OUT	Description
schemaURL	(IN)	URL identifying the schema.

generateSchema()

Description

These functions generate XML schema(s) from an oracle type name.   Can result in a `ORA-31001` exception: invalid resource handle or path name. The available options are given in the following table.

Syntax	Description
function generateSchemas(schemaName IN varchar2, typeName IN varchar2, elementName IN varchar2 := NULL, schemaURL IN varchar2 := NULL, annotate IN BOOLEAN := TRUE, embedColl IN BOOLEAN := TRUE) return sys.XMLSequenceType;	Returns a collection of XMLTypes, one XMLSchema document for each database schema.

Syntax	Description

function generateSchema(
    schemaName IN varchar2,
    typeName IN varchar2,
    elementName IN varchar2 := NULL,
    recurse IN BOOLEAN := TRUE,
    annotate IN BOOLEAN := TRUE,
    embedColl IN BOOLEAN := TRUE)
 return sys.XMLType;
	Inlines all in one schema (XMLType).

Parameter	IN / OUT	Description
schemaName	(IN)	Name of the database schema containing the type.
typeName	(IN)	Name of the oracle type.
elementName	(IN)	The name of the toplevel element in the XMLSchema defaults to typeName.
schemaURL	(IN)	Dpecifies base URL where schemas will be stored, needed by top level schema for import statement.
recurse	(IN)	Whether or not to also generate schema for all types referred to by the type specified.
annotate	(IN)	Whether or not to put the SQL annotations in the XMLSchema.
embedColl	(IN)	Should the collections be embedded in the type which refers to them, or create a complexType? Cannot be FALSE if annotations are turned on.

Catalog Views

Table 28–3 Summary of Catalog View Schemas

Schema	Description
USER_XML_SCHEMAS on page 28-10	All registered XML Schemas owned by the user.
ALL_XML_SCHEMAS on page 28-11	All registered XML Schemas usable by the current user.
DBA_XML_SCHEMAS on page 28-11	All registered XML Schemas in Oracle XML DB.
DBA_XML_TABLES on page 28-11	All XMLType tables in the system.
USER_XML_TABLES on page 28-12	All XMLType tables owned by the current user.
ALL_XML_TABLES on page 28-12	All XMLType tables usable by the current user.
DBA_XML_TAB_COLS on page 28-13	All XMLType table columns in the system.
USER_XML_TAB_COLS on page 28-13	All XMLType table columns in tables owned by the current user.
ALL_XML_TAB_COLS on page 28-13	All XMLType table columns in tables usable by the current user.
DBA_XML_VIEWS on page 28-14	All XMLType views in the system.
USER_XML_VIEWS on page 28-14	All XMlType views owned by the current user.
ALL_XML_VIEWS on page 28-15	All XMLType views usable by the current user.
DBA_XML_VIEW_COLS on page 28-15	All XMLType view columns in the system.
USER_XML_VIEW_COLS on page 28-15	All XMLType view columns in views owned by the current user.
ALL_XML_VIEW_COLS on page 28-16	All XMLType view columns in views usable by the current user.

USER_XML_SCHEMAS

Description

Lists all schemas (local and global) belonging to the current user.

Column	Datatype	Description
SCHEMA_URL	VARCHAR2	URL of XML schema
LOCAL	VARCHAR2	Local schema (YES / NO)

Column	Datatype	Description
SCHEMA	XMLTYPE	XML Schema document

ALL_XML_SCHEMAS

Description

Lists all local schemas belonging to the current user and all global schemas.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning XML schema
SCHEMA_URL	VARCHAR2	URL of XML schema
LOCAL	VARCHAR2	Local schema (YES / NO)
SCHEMA	XMLTYPE	XML Schema document

DBA_XML_SCHEMAS

Description

Lists all registered local and global schemas in the system.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning XML schema
SCHEMA_URL	VARCHAR2	URL of XML schema
LOCAL	VARCHAR2	Local schema (YES / NO)
SCHEMA	XMLTYPE	XML Schema document

DBA_XML_TABLES

Description

Lists all XMLType tables in the system.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning table

Column	Datatype	Description
TABLE_NAME	VARCHAR2	Name of XMLType table
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

USER_XML_TABLES

Description

Lists all local XMLType tables belonging to the current user.

Column	Datatype	Description
TABLE_NAME	VARCHAR2	Name of XMLType table
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

ALL_XML_TABLES

Description

Lists all local XMLType tables belonging to the current user and all global tables visible to the current user.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning table
TABLE_NAME	VARCHAR2	Name of XMLType table
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

DBA_XML_TAB_COLS

Description

Lists all XMLType columns in the system.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning table
TABLE_NAME	VARCHAR2	Name of table
COLUMN_NAME	VARCHAR2	Name of XMLType column
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

USER_XML_TAB_COLS

Description

Lists all XMLType columns in tables belonging to the current user.

Column	Datatype	Description
TABLE_NAME	VARCHAR2	Name of table
COLUMN_NAME	VARCHAR2	Name of XMLType column
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

ALL_XML_TAB_COLS

Description

Lists all XMLType columns in tables belonging to the current user and all global tables visible to the current user.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning table
TABLE_NAME	VARCHAR2	Name of table
COLUMN_NAME	VARCHAR2	Name of XMLType column
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element
STORAGE_TYPE	VARCHAR2	Storage type: CLOB / OBJECT-RELATIONAL

DBA_XML_VIEWS

Description
Lists all XMLType views in the system.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning view
VIEW_NAME	VARCHAR2	Name of XMLType view
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element

USER_XML_VIEWS

Description
Lists all local XMLType views belonging to the current user.

Column	Datatype	Description
VIEW_NAME	VARCHAR2	Name of XMLType view
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element

ALL_XML_VIEWS

Description

Lists all local XMLType views belonging to the current user and all global views visible to the current user.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning view
VIEW_NAME	VARCHAR2	Name of XMLType view
XMLSCHEMA	VARCHAR2	XML Schema URL
ELEMENT_NAME	VARCHAR2	XML Schema element

DBA_XML_VIEW_COLS

Description

Lists all XMLType columns in the system.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning view.
VIEW_NAME	VARCHAR2	Name of view.
COLUMN_NAME	VARCHAR2	Name of XMLType column.
XMLSCHEMA	VARCHAR2	XML Schema URL.
ELEMENT_NAME	VARCHAR2	XML Schema element.

USER_XML_VIEW_COLS

Description

Lists all XMLType columns in views belonging to the current user.

Column	Datatype	Description
VIEW_NAME	VARCHAR2	Name of view.
COLUMN_NAME	VARCHAR2	Name of XMLType column.

Column	Datatype	Description
XMLSCHEMA	VARCHAR2	XML Schema URL.
ELEMENT_NAME	VARCHAR2	XML Schema element.

ALL_XML_VIEW_COLS

Description

Lists all XMLType columns in views belonging to the current user and all global views visible to the current user.

Column	Datatype	Description
OWNER	VARCHAR2	Database user owning view.
VIEW_NAME	VARCHAR2	Name of view.
COLUMN_NAME	VARCHAR2	Name of XMLType column.
XMLSCHEMA	VARCHAR2	XML Schema URL.
ELEMENT_NAME	VARCHAR2	XML Schema element.

29

Resource Management and Access Control for PL/SQL

Resource Management and Access Control APIs for PL/SQL are contained in the DBMS_XDB Package.

This chapter contains the following sections:

- Description of DBMS_XDB
- Functions and Procedures of DBMS_XDB

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

DBMS_XDB Package

Description of DBMS_XDB

The DBMS_XDB package provides the PL/SQL application developer with APIs that allow resource management in the Oracle XML DB Hierarchy, support for Oracle XML DB's Access Control List (ACL) Security and Oracle XML DB Configuration sessional management.

The Oracle XML DB Resource Management functionality provides `Link()`, `LockResource()`, `GetLockToken()`, `UnlockResource()`, `CreateResource()`, `CreateFolder()`, `DeleteResource()`,`Link()` and functions. These methods complement the functionality provided by Resource Views.

The ACL-based security mechanism can be used with either in-hierarchy ACLs (ACLs stored by the Oracle XML DB resource API) or in-memory ACLs (that may be stored by the user outside Oracle XML DB). Some of these methods can be used for both Oracle XML DB resources and arbitrary database objects.

The Access Control Security functionality provides `checkPrivileges()`, `getAclDocument()`, `changePrivileges()` and `getPrivileges()` functions for Oracle XML DB Resources. `AclCheckPrivileges()` function enables database users access to Oracle XML DB's ACL-based Security mechanism without having to have their objects stored in the Oracle XML DB Hierarchy.

Oracle XML DB Configuration session management provides `CFG_Refresh()`, `CFG_Get()` and `CFG_Update()`.

Functions and Procedures of DBMS_XDB

Table 29–1 Summary of Functions and Procedures of DBMS_XDB

Function/Procedure	Description
getAclDocument() on page 29-3	Retrieves ACL document that protects resource given its path name.
getPrivileges() on page 29-4	Gets all privileges granted to the current user on the given Oracle XML DB resource.
changePrivileges() on page 29-4	Adds the given ACE to the given resource's ACL.
checkPrivileges() on page 29-5	Checks access privileges granted to the current user on the specified Oracle XML DB resource.

Table 29–1 Summary of Functions and Procedures of DBMS_XDB (Cont.)

Function/Procedure	Description
setacl() on page 29-6	Sets the ACL on the given Oracle XML DB resource to be the ACL specified.
AclCheckPrivileges() on page 29-6	Checks access privileges granted to the current user by specified ACL document on a resource whose owner is specified by the 'owner' parameter.
LockResource() on page 29-7	Gets a WebDAV-style lock on that resource given a path to that resource.
GetLockToken() on page 29-8	Returns that resource's lock token for the current user given a path to a resource.
UnlockResource() on page 29-8	Unlocks the resource given a lock token and a path to the resource.
CreateResource() on page 29-9	Creates a new resource.
CreateFolder() on page 29-10	Creates a new folder resource in the hierarchy.
DeleteResource() on page 29-10	Deletes a resource from the hierarchy.
Link() on page 29-11	Creates a link to an existing resource.
CFG_Refresh() on page 29-11	Refreshes the session's configuration information to the latest configuration.
CFG_Get() on page 29-11	Retrieves the session's configuration information.
CFG_Update() on page 29-12	Updates the configuration information.

getAclDocument()

Description

Retrieves ACL document that protects resource given its path name; returns the `xmltype` for ACL document.

Syntax

```
FUNCTION getAclDocument( abspath  IN  VARCHAR2)
                         RETURN sys.xmltype;
```

Parameter	IN / OUT	Description
abspath	(IN)	Pathname of the resource whose ACL doc is required.

getPrivileges()

Description

Gets all privileges granted to the current user on the given Oracle XML DB resource. Returns an XMLType instance of <privilege> element, which contains the list of all leaf privileges granted on this resource to the current user. For example,

```
<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
                              http://xmlns.oracle.com/xdb/acl.xsd"
      <read-contents/>
      <read-properties/>
      <resolve/>
      <read-acl/>
 </privilege>
```

Syntax

```
FUNCTION getPrivileges( res_path IN VARCHAR2) RETURN sys.xmltype;
```

Parameter	IN / OUT	Description
res_path	(IN)	Absolute path in the Hierarchy of the Oracle XML DB resource.

changePrivileges()

Description

Adds the given ACE to the given resource's ACL. Returns positive integer if ACL was successfully modified. For example,

```
<ace  xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:dav="DAV:"
          xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
                              http://xmlns.oracle.com/xdb/acl.xsd
                              DAV:http://xmlns.oracle.com/xdb/dav.xsd"
     <grant>true</grant>
     <principal>SCOTT</principal>
     <privilege>
          <read-contents/>
```

```
                <read-properties/>
                <resolve/>
                <dav:waste/>
            </privilege>
        </ace>
```

Syntax

```
FUNCTION changePrivileges( res_path  IN  VARCHAR2,
                           ace       IN  xmltype)
                           RETURN pls_integer;
```

Parameter	IN / OUT	Description
res_path	(IN)	Pathname of the Oracle XML DB resource for which privileges need to be changed.
ace	(IN)	An XMLType instance of the `<ace>` element which specifies the `<principal>`, the operation `<grant>` and the list of privileges. See the preceding code example.

If no ACE with the same principal and the same operation (`grant`/`deny`) already exists in the ACL, the new ACE is added at the end of the ACL.

checkPrivileges()

Description

Checks access privileges granted to the current user on the specified Oracle XML DB resource. Returns positive integer if all requested privileges granted. For example, check for `<read.contents>`, `<read.properties>` and `<dav:waste>` privileges using the following `<privilege>` XMLType instance.

```
<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:dav="DAV:"
           xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
                               http://xmlns.oracle.com/xdb/acl.xsd
                               DAV: http://xmlns.oracle.com/xdb/dav.xsd"
     <read-contents/>
     <read-properties/>
     <resolve/>
     <dav:waste/>
</privilege>
```

Syntax

```
FUNCTION checkPrivileges( res_path   IN  VARCHAR2,
                          privs      IN  xmltype)
                          RETURN pls_integer;
```

Parameter	IN / OUT	Description
res_path	(IN)	Absolute path in the Hierarchy for Oracle XML DB resource.
privs	(IN)	An XMLType instance of the privilege element specifying the requested set of access privileges. See the preceding code example.

setacl()

Description

Sets the ACL on the given Oracle XML DB resource to be the ACL specified by path. The user must have `<write-acl>` privileges on the resource.

Syntax

```
PROCEDURE setacl( res_path   IN  VARCHAR2,
                 acl_path   IN  VARCHAR2);
```

Parameter	IN / OUT	Description
res_path	(IN)	Absolute path in the Hierarchy for Oracle XML DB resource.
acl_path	(IN)	Absolute path in the Hierarchy for Oracle XML DB ACL.

AclCheckPrivileges()

Description

Checks access privileges granted to the current user by specified ACL document on a resource whose owner is specified by the 'owner' parameter. Returns positive integer if all requested privileges granted.

Syntax

```
FUNCTION AclCheckPrivileges( acl_path  IN  VARCHAR2,
                             owner     IN  VARCHAR2,
                 privs     IN  xmltype)
                 RETURN pls_integer;
```

Parameter	IN / OUT	Description
acl_path	(IN)	Absolute path in the Hierarchy for ACL document.
owner	(IN)	Resource owner name; the pseudo user "DAV:owner" is replaced by this user during ACL privilege resolution.
privs	(IN)	An XMLType instance of the privilege element specifying the requested set of access privileges. See description for `checkPrivileges()checkPrivileges()`..

LockResource()

Description

Given a path to a resource, gets a WebDAV-style lock on that resource. Returns `TRUE` if operation successful; `FALSE`, otherwise. The user must have `UPDATE` privileges on the resource.

Syntax

```
FUNCTION LockResource( path      IN  VARCHAR2,
                       depthzero IN  BOOLEAN,
                       shared    IN  boolean)
                       RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
path	(IN)	Path name of the resource to lock.
depthzero	(IN)	CURRENTLY UNSUPPORTED. At this time, only the given resource is locked by this function. In a future release, passing FALSE will obtain an infinite-depth lock.
shared	(IN)	Passing TRUE will obtain a shared write lock.

GetLockToken()

Description

Given a path to a resource, returns that resource's lock token for the current user. The user must have READPROPERTIES privilege on the resource.

Syntax

```
PROCEDURE GetLockToken( path      IN  VARCHAR2,
                        locktoken OUT VARCHAR2);
```

Parameter	IN / OUT	Description
path	(IN)	Path name to the resource.
locktoken	(OUT)	Logged-in user’s lock token for the resource.

UnlockResource()

Description

Unlocks the resource given a lock token and a path to the resource. Returns TRUE if operation successful; FALSE, otherwise. The user must have UPDATE privileges on the resource.

Syntax

```
FUNCTION UnlockResource( path     IN  VARCHAR2,
                         deltoken IN  VARCHAR2)
                         RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
path	(IN)	Path name to the resource.
deltoken	(IN)	Lock token to be removed.

CreateResource()

Description

Creates a new resource. Returns TRUE if operation successful; FALSE, otherwise.The options are described in the following table.

Syntax	Description
FUNCTION CreateResource(path IN VARCHAR2, data IN VARCHAR2) RETURN BOOLEAN;	Creates a new resource with the given string as its contents.
FUNCTION CreateResource(path IN VARCHAR2, data IN SYS.XMLTYPE) RETURN BOOLEAN;	Creates a new resource with the given XMLType data as its contents.
FUNCTION CreateResource(path IN VARCHAR2, datarow IN REF SYS.XMLTYPE) RETURN BOOLEAN;	Given a REF to an existing XMLType row, creates a resource whose contents point to that row. That row should not already exist inside another resource.
FUNCTION CreateResource(path IN VARCHAR2, data IN CLOB) RETURN BOOLEAN;	Creates a resource with the given CLOB as its contents.
FUNCTION CreateResource(path IN VARCHAR2, data IN BFILE) RETURN BOOLEAN;	Creates a resource with the given BFILE as its contents.

Parameter	IN / OUT	Description
path	(IN)	Path name of the resource to create. The path name's parent folder must already exist in the hierarchy. In other words, if '/foo/bar.txt' is passed in, then folder '/foo' must already exist.

Parameter	IN / OUT	Description
data	(IN)	The new resource’s contents. The data will be parsed to check if it contains a schema-based XML document, and the contents will be stored as schema-based in the schema's default table. Otherwise, it will be saved as binary data.
datarow	(IN)	REF to an XMLType row to be used as the contents.

CreateFolder()

Description

Creates a new folder resource in the hierarchy. Returns `TRUE` if operation successful; `FALSE`, otherwise. The given path name's parent folder must already exist in the hierarchy; for example, if `'/folder1/folder2'` is passed as the path parameter, then `'/folder1'` must already exist.

Syntax

```
FUNCTION CreateFolder( path   IN  VARCHAR2)
                       RETURN BOOLEAN;
```

Parameter	IN / OUT	Description
path	(IN)	Path name for the new folder.

DeleteResource()

Description

Deletes a resource from the hierarchy.

Syntax

```
PROCEDURE DeleteResource( path   IN  VARCHAR2);
```

Parameter	IN / OUT	Description
path	(IN)	Path name of the resource to delete.

Link()

Description

Creates a link to an existing resource. This procedures is analogous to creating a hard link in UNIX.

Syntax

```
PROCEDURE Link( srcpath    IN  VARCHAR2,
                linkfolder IN  VARCHAR2,
                linkname   IN  VARCHAR2);
```

Parameter	IN / OUT	Description
srcpath	(IN)	Path name of the resource to which a link is made
linkfolder	(IN)	Folder in which the new link is placed.
linkname	(IN)	Name of the new link.

CFG_Refresh()

Description

Refreshes the session's configuration information to the latest configuration.

Syntax

```
PROCEDURE CFG_Refresh;
```

CFG_Get()

Description

Retrieves the session's configuration information as an XMLType instance.

Syntax

```
FUNCTION CFG_Get RETURN SYS.XMLType;
```

CFG_Update()

Description

Updates the configuration information and commits the change.

Syntax

```
PROCEDURE CFG_Update( xdbconfig   IN  SYS.XMLTYPE);
```

Parameter	IN / OUT	Description
xdbconfig	(IN)	The new configuration data.

30

Generating Queries Using DBMS_XMLGEN for PL/SQL

The DBMS_XMLGEN Package is used to transform results of SQL queries into a canonical XML format.

This chapter discusses the following topics:

- Description of DMS_XMLGEN
- Functions and Procedures of DBMS_XMLGEN

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

DBMS_XMLGEN Package

Description of DMS_XMLGEN

DBMS_XMLGEN converts the results of a SQL query to a canonical XML format. The package takes an arbitrary SQL query as input, converts it to XML format, and returns the result as a CLOB.

This package is similar to the DBMS_XMLQUERY package, except that it is written in C and compiled into the kernel. This package can only be run in the database.

Functions and Procedures of DBMS_XMLGEN

Table 30–1 Summary of Functions and Procedures of DBMS_XMLGEN

Function/Procedure	Description
newContext() on page 30-3	Creates a new context handle.
setRowTag() on page 30-3	Sets the name of the element enclosing each row of the result. The default tag is ROW.
setRowSetTag () on page 30-4	Sets the name of the element enclosing the entire result. The default tag is ROWSET.
getXML() on page 30-4	Gets the XML document.
getNumRowsProcessed() on page 30-6	Gets the number of SQL rows that were processed in the last call to getXML.
setMaxRows() on page 30-6	Sets the maximum number of rows to be fetched each time.
setSkipRows() on page 30-7	Sets the number of rows to skip every time before generating the XML. The default is 0.
setConvertSpecialChars() on page 30-7	Sets whether special characters such as $, which are non-XML characters, should be converted or not to their escaped representation. The default is to perform the conversion.
convert() on page 30-8	Converts the XML into the escaped or unescaped XML equivalent.
useItemTagsForColl() on page 30-9	Forces the use of the collection column name appended with the tag _ITEM for collection elements. The default is to set the underlying object type name for the base element of the collection.
restartQUERY() on page 30-9	Restarts the query to start fetching from the beginning.

Table 30–1 Summary of Functions and Procedures of DBMS_XMLGEN (Cont.)

Function/Procedure	Description
closeContext() on page 30-9	Closes the context and releases all resources.

newContext()

Description

Generates and returns a new context handle; this context handle is used in `getXML()` and other functions to get XML back from the result. The available options are given in the following table.

Syntax	Description
DBMS_XMLGEN.newContext (query IN VARCHAR2) RETURN ctxHandle;	Generates a new context handle from a query.
DBMS_XMLGEN.newContext (queryString IN SYS_REFCURSOR) RETURN ctxHandle;	Generates a new context handle from a query string in the form of a PL/SQL ref cursor

Parameter	IN / OUT	Description
query	(IN)	The query, in the form of a `VARCHAR`, the result of which must be converted to XML
queryString	(IN)	The query string in the form of a PL/SQL ref cursor, the result of which must be converted to XML.

setRowTag()

Description

Sets the name of the element separating all the rows. The default name is `ROW.` User can set this to `NULL` to suppress the `ROW` element itself. However, an error is produced if both the row and the rowset are `NULL` and there is more than one column or row in the output; this is because the generated XML would not have a top-level enclosing tag, and so would be invalid.

Syntax

```
DBMS_XMLGEN.setRowTag (
   ctx     IN ctxHandle,
   rowTag  IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle obtained from the `newContext` call.
rowTag	(IN)	The name of the `ROW` element. Passing `NULL` indicates that you do not want the `ROW` element present.

setRowSetTag ()

Description

Sets the name of the root element of the document. The default name is `ROWSET.` User can set the `rowSetTag` `NULL` to suppress the printing of this element. However, an error is produced if both the row and the rowset are `NULL` and there is more than one column or row in the output; this is because the generated XML would not have a top-level enclosing tag, and so would be invalid.

Syntax

```
DBMS_XMLGEN.setRowSetTag (
   ctx        IN ctxHandle,
   rowSetTag  IN VARCHAR2);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle obtained from the `newContext` call.
rowSetTag	(IN)	The name of the document element. Passing `NULL` indicates that you do not want the `ROWSET` element present.

getXML()

Description

Gets the XML document. When the rows indicated by the `setSkipRows()` call are skipped, the maximum number of rows as specified by the `setMaxRows()` call (or the entire result if not specified) is fetched and converted to XML. Use the `getNumRowsProcessed()`

to check if any rows were retrieved. The available options are given in the following table.

Syntax	Description
FUNCTION DBMS_XMLGEN.getXML (ctx IN ctxHandle, clobval IN OUT NCOPY clob, dtdOrSchema IN number := NONE) RETURN boolean;	This procedure gets the XML document by fetching the maximum number of rows specified. It appends the XML document to the CLOB passed in. Use this version of getXML() to avoid any extra CLOB copies and to reuse the same CLOB for subsequent calls. Because of the CLOB reuse, this getXML() call is potentially more efficient.
FUNCTION DBMS_XMLGEN.getXML (ctx IN ctxHandle, dtdOrSchema IN number := NONE) RETURN clob;	Generates the XML document and returns it as a temporary CLOB. The temporary CLOB obtained from this function must be freed using the `DBMS_LOB.FREETEMPORARY` call.
FUNCTION DBMS_XMLGEN.getXML (sqlQuery IN VARCHAR2, dtdOrSchema IN number := NONE) RETURN clob;	Converts the results from the SQL query string to XML format, and returns the XML as a temporary CLOB. This temporary CLOB must be subsequently freed using the `DBMS_LOB.FREETEMPORARY` call.
FUNCTION DBMS_XMLGEN.getXMLType (ctx IN ctxhandle, dtdOrSchema IN number := NONE) RETURN sys.XMLType;	Generates the XML document and returns it as a `sys.XMLType`. `XMLType` operations can be performed on the results, including `ExistsNode` and `Extract`. This also provides a way of obtaining the results as a string by using the `getStringVal()` function, if the result size is less than 4K.
FUNCTION DBMS_XMLGEN.getXMLType (sqlQuery IN VARCHAR2, dtdOrSchema IN number := NONE) RETURN sys.XMLType	Converts the results from the SQL query string to XML format, and returns the XML as a `sys.XMLType`. `XMLType` operations can be performed on the results, including `ExistsNode` and `Extract`. This also provides a way of obtaining the results as a string by using the `getStringVal()` function, if the result size is less than 4K.

Parameter	IN / OUT	Description
ctx	(IN)	The context handle obtained from the `newContext` call.
clobval	(IN/OUT)	The clob to which the XML document is appended.
sqlQuery	(IN)	The SQL query string.

Parameter	IN / OUT	Description
dtdOrSchema	(IN)	The Boolean to indicate generation of either a DTD or a schema. NONE is the only option currently supported.

getNumRowsProcessed()

Description

Retrieves the number of SQL rows processed when generating the XML using the `getXML` call; this count does not include the number of rows skipped before generating the XML. Used to determine the terminating condition if calling `getXML()` in a loop. Note that `getXML()` always generates an XML document, even if there are no rows present.

Syntax

```
DBMS_XMLGEN.getNumRowsProcessed (
   ctx  IN ctxHandle)
RETURN NUMBER;
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle obtained from the `newContext` call.

setMaxRows()

Description

Sets the maximum number of rows to fetch from the SQL query result for every invokation of the `getXML` call. Used when generating paginated results. For example, when generating a page of XML or HTML data, restrict the number of rows converted to XML or HTML by setting the `maxRows` parameter.

Syntax

```
DBMS_XMLGEN.setMaxRows (
   ctx      IN ctxHandle,
   maxRows  IN NUMBER);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle corresponding to the query executed.
maxRows	(IN)	The maximum number of rows to get for each call to `getXML`.

setSkipRows()

Description

Skips a given number of rows before generating the XML output for every call to the `getXML` routine. Used when generating paginated results for stateless Web pages using this utility. For example, when generating the first page of XML or HTML data, set `skipRows` to zero. For the next set, set the `skipRows` to the number of rows obtained in the first case. See `getNumRowsProcessed()`.

Syntax

```
DBMS_XMLGEN.setSkipRows (
   ctx        IN ctxHandle,
   skipRows   IN NUMBER);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle corresponding to the query executed.
skipRows	(IN)	The number of rows to skip for each call to `getXML`.

setConvertSpecialChars()

Description

Sets whether or not special characters in the XML data must be converted into their escaped XML equivalent. For example, the < sign is converted to `&lt;`. The default is to perform conversions. Improves performance of XML processing when the input data cannot contain any special characters such as <, >, `"`, `'`, which must be escaped. It is expensive to scan the character data to replace the special characters, particularly if it involves a lot of data. Syntax

```
DBMS_XMLGEN.setConvertSpecialChars (
   ctx    IN ctxHandle,
   conv   IN boolean);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle obtained from the `newContext` call.
conv	(IN)	`TRUE` indicates that conversion is needed.

convert()

Description

Converts the XML data into the escaped or unescaped XML equivalent; returns XML CLOB data in encoded or decoded format. Escapes the XML data if the `ENTITY_ENCODE` is specified. For example, the escaped form of the character < is `&lt;`. Unescaping is the reverse transformation. The available options are given in the following table.

Syntax	Description

DBMS_XMLGEN.convert (
  xmlData IN VARCHAR2,
  flag   IN NUMBER := ENTITY_ENCODE)
RETURN VARCHAR2;
	Uses xmlData in string form (VARCHAR2).

DBMS_XMLGEN.convert (
  xmlData IN CLOB,
  flag   IN NUMBER := ENTITY_ENCODE)
RETURN CLOB;
	Uses xmlData in Clob form.

Parameter	IN / OUT	Description
xmlData	(IN)	The XML CLOB data to be encoded or decoded.
flag	(IN)	The flag setting; `ENTITY_ENCODE` (default) for encode, and `ENTITY_DECODE` for decode.

useItemTagsForColl()

Description

Overrides the default name of the collection elements. The default name for collection elements is the type name itself. Using this function, you can override the default to use the name of the column with the `_ITEM` tag appended to it. If there is a collection of `NUMBER`, the default tag name for the collection elements is `NUMBER`. Using this procedure, the user can override this behavior and generate the collection column name with the `_ITEM` tag appended to it.

Syntax

```
DBMS_XMLGEN.useItemTagsForColl (
   ctx  IN ctxHandle);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle.

restartQUERY()

Description

Restarts the query and generates the XML from the first row. Can be used to start executing the query again, without having to create a new context.

Syntax

```
DBMS_XMLGEN.restartQUERY (ctx  IN ctxHandle);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle corresponding to the current query.

closeContext()

Description

Closes a given context and releases all resources associated with it, including the SQL cursor and bind and define buffers. After this call, the handle cannot be used for a subsequent `DBMS_XMLGEN` function call.

Syntax

```
DBMS_XMLGEN.closeContext ( ctx  IN ctxHandle);
```

Parameter	IN / OUT	Description
ctx	(IN)	The context handle to close.

31

Oracle XML DB Resource View API for PL/SQL

This chapter contains the following sections:

- Description of Oracle XML DB Resource View package
- Operators of Oracle XML DB Resource View package

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*

Oracle XML DB Resource View API

Description of Oracle XML DB Resource View package

The resource view and path view both provide a mechanism for SQL access for data that is stored in the Oracle XML DB repository. Data which is stored in the Oracle XML DB repository ia protocols like FTP, WebDAV or programming API such as JNDI can be accessed in SQL through these views and vice versa. Resource view and path view together (along with some PL/SQL packages) provide all the query and DML functionality that is available through the programming API. The PATH VIEW has one row for each unique path in the repository, whereas the resource view has one row for each resource in the repository.

Operators of Oracle XML DB Resource View package

Table 31–1 Summary of Operators of Oracle XML DB Resource View package

Operator	Description
UNDER_PATH on page 31-2	Using the Oracle XML DB hierarchical index, returns sub-paths of a particular path.
EQUALS_PATH on page 31-3	Finds the resource with the specified path name.
PATH on page 31-4	Returns the relative path name of the resource under the specified path name argument.
DEPTH on page 31-4	Returns the folder depth of the resource under the specified starting path.

UNDER_PATH

Description

The `UNDER_PATH` operator uses the Oracle XML DB hierarchical index to return the paths under a particular path. The hierarchical index is designed to speed access walking down a path name (the normal usage). If the other parts of the query predicate are very selective, however, a functional implementation of `UNDER_PATH` `may` be chosen that will walk back up the repository. This can be more efficient, since a much smaller number of links may need to be traversed. The options are described in the following table.

Syntax	Description
INTEGER UNDER_PATH(resource_column, pathname);	Determines if a resource is under a specified path.
INTEGER UNDER_PATH(resource_column, depth, pathname);	Determines if a resource is under a specified path, with a depth argument to restrict the number of levels to search.
INTEGER UNDER_PATH(resource_column, pathname, correlation)	Determines if a resource is under a specified path, with a correlation argument for ancillary operators.
INTEGER UNDER_PATH(resource_column, depth, pathname, correlation)	Determines if a resource is under a specified path with a depth argument to restrict the number of levels to search, and with a correlation argument for ancillary operators. Note that only one of the accessible paths to the resource needs to be under the path argument for a resource to be returned.

Parameter	Description
resource_column	The column name or column alias of the 'resource' column in the `path_view` or `resource_view`.
pathname	The path name to resolve.
depth	The maximum depth to search; a depth of less than `0` is treated as `0`.
correlation	An integer that can be used to correlate the `UNDER_PATH` operator (a primary operator) with ancillary operators (`PATH` & `DEPTH`).

EQUALS_PATH

Description

Finds the resource with the specified path name. The `EQUALS_PATH` operator is functionally equivalent to `UNDER_PATH` with a depth restriction of `0`.

Syntax

```
EQUALS_PATH INTEGER EQUALS_PATH( resource_column,
                                 pathname);
```

Parameter	Description
resource_column	The column name or column alias of the 'resource' column in the `path_view` or `resource_view`.
pathname	The path name to resolve.

PATH

Description

An ancillary operator that returns the relative path name of the resource under the specified `pathname` argument. Note that the path column in the resource view always contains the absolute path of the resource.

Syntax

```
PATH VARCHAR2 PATH( correlation);
```

Parameter	Description
correlation	An integer that can be used to correlate the UNDER_PATH operator (a primary operator) with ancillary operators (PATH & DEPTH).

DEPTH

Description

An ancillary operator that returns the folder depth of the resource under the specified starting path.

Syntax

```
DEPTH  INTEGER DEPTH( correlation);
```

Parameter	Description
correlation	An integer that can be used to correlate the UNDER_PATH operator (a primary operator) with ancillary operators (PATH & DEPTH).

32

Oracle XML DB Versioning API for PL/SQL

Oracle XML DB Versioning APIs are found in the DBMS_XDB_VERSION Package.

This chapter contains the following sections:

- Description of DBMS_XDB_VERSION
- Functions and Procedures of DBMS_XDB_VERSION

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

DBMS_XDB_VERSION Package

Description of DBMS_XDB_VERSION

Functions and procedures of DBMS_XDB_VERSION help to create a VCR and manage the versions in the version history.

Functions and Procedures of DBMS_XDB_VERSION

Table 32–1 Summary of Functions and Procedures of DBMS_XDB_VERSION

Function/Procedure	Description
MakeVersioned() on page 32-2	Turns a regular resource whose path name is given into a version-controlled resource.
Checkout() on page 32-3	Checks out a VCR before updating or deleting it.
Checkin() on page 32-3	Checks in a checked-out VCR and returns the resource id of the newly-created version.
Uncheckout() on page 32-4	Checks in a checked-out resource and returns the resource id of the version before the resource is checked out.
GetPredecessors() on page 32-4	Retrieves the list of predecessors by path name.
GetPredsByResId() on page 32-5	Retrieves the list of predecessors by resource id.
GetResourceByResId() on page 32-5	Obtains the resource as an XMLType, given the resource objectID.
GetSuccessors() on page 32-5	Retrieves the list of successors by path name.
GetSuccsByResId() on page 32-6	Retrieves the list of successors by resource id.

MakeVersioned()

Description

Turns a regular resource whose path name is given into a version-controlled resource. If two or more path names are bound with the same resource, a copy of the resource will be created, and the given path name will be bound with the newly-created copy. This new resource is then put under version control. All other path names continue to refer to the original resource. This function returns the resource ID of the first version, or root, of the VCR. This is not an auto-commit SQL operation.

- It is legal to call `MakeVersioned()` for VCR, and neither exception nor warning is raised.
- It is illegal to call `MakeVersioned()` for folder, version history, version resource, and ACL.
- No support for Schema-based resources is provided.

An exception is raised if the resource doesn't exist.

Syntax

```
FUNCTION MakeVersioned( pathname VARCHAR2) RETURN dbms_xdb.resid_type;
```

Parameter	Description
pathname	The path name of the resource to be put under version control.

Checkout()

Description

Checks out a VCR before updating or deleting it. This is not an auto-commit SQL operation. Two users of the same workspace cannot `Checkout()` the same VCR at the same time. If this happens, one user must rollback. As a result, it is good practice to commit the `Checkout()` operation before updating a resource and avoid loss of the update if the transaction is rolled back. An exception is raised if the given resource is not a VCR, if the VCR is already checked out, if the resource doesn't exist.

Syntax

```
PROCEDURE Checkout(  pathname VARCHAR2);
```

Parameter	Description
pathname	The path name of the VCR to be checked out.

Checkin()

Description

Checks in a checked-out VCR and returns the resource id of the newly-created version. This is not an auto-commit SQL operation. `Checkin()` doesn't have to take the same path name that was passed to `Checkout()` operation. However, the `Checkin()` path name and the `Checkout()` path name must be of the same resource for the operations to function correctly. If the resource has been renamed, the new name must be used to `Checkin()` because the old name is either invalid or is currently bound with a different resource. Exception is raised if the path name doesn't exist. If the path name has been changed, the new path name must be used to `Checkin()` the resource.

Syntax

```
FUNCTION Checkin( pathname VARCHAR2) RETURN dbms_xdb.resid_type;
```

Parameter	Description
pathname	The path name of the checked-out resource.

Uncheckout()

Description

Checks in a checked-out resource and returns the resource id of the version before the resource is checked out. This is not an auto-commit SQL operation. `Uncheckout()` doesn't have to take the same path name that was passed to `Checkout()` operation. However, the `Uncheckout()` path name and the `Checkout()` path name must be of the same resource for the operations to function correctly. If the resource has been renamed, the new name must be used to uncheckout because the old name is either invalid or is currently bound with a different resource. An exception is raised if the path name doesn't exist. If the path name has been changed, the new path name must be used to `Checkin()` the resource.

Syntax

```
FUNCTION Uncheckout( pathname VARCHAR2) RETURN dbms_xdb.resid_type;
```

Parameter	Description
pathname	The path name of the checked-out resource.

GetPredecessors()

Description

Retrieves the list of predecessors by path name. An exception is raised if the `pathname` is illegal.

Syntax

```
FUNCTION GetPredecessors( pathname VARCHAR2) RETURN resid_list_type;
```

Parameter	Description
pathname	The path name of the resource.

GetPredsByResId()

Description

Retrieves the list of predecessors by resource id. Getting predecessors by `resid` is more efficient than by path name. An exception is raised if the `resid` is illegal.

Syntax

```
FUNCTION GetPredsByResId( resid resid_type) RETURN resid_list_type;
```

Parameter	Description
resid	The resource id.

GetResourceByResId()

Description

Obtains the resource as an XMLType, given the resource objectID. Because the system will not create a path name for versions, this function is useful for retrieving the resource using its resource id.

Syntax

```
FUNCTION GetResourceByResId( resid resid_type) RETURN XMLType;
```

Parameter	Description
resid	The resource id.

GetSuccessors()

Description

Given a version resource or a VCR, retrieves the list of the successors of the resource by path name. Getting successors by `resid` is more efficient than by `pathname`. An exception is raised if the `pathname` is illegal.

Syntax

```
FUNCTION GetSuccessors( pathname VARCHAR2) RETURN resid_list_type;
```

Parameter	Description
pathname	The path name of the resource

GetSuccsByResId()

Description

Given a version resource or a VCR, retrieves the list of the successors of the resource by resource id. Getting successors by resid is more efficient than by path name. An exception is raised if the resid is illegal.

Syntax

```
FUNCTION GetSuccsByResId( resid resid_type) RETURN resid_list_type;
```

Parameter	Description
resid	The resource id.

33 Managing ConText Indexes: DBMS_XDBT in PL/SQL

The DBMS_XDBT Package enables an administrator to create a ConText index on the XML DB hierarchy and configure it for automatic maintenance.

This chapter contains the following sections:

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

DBMS_XDBT Package

Description of DBMS_XDBT

The DBMS_XDBT package provides a convenient mechanism for administrators to set up a ConText index on the Oracle XML DB Hierarchy. The package contains procedures to create default preferences, create the index and set up automatic synchronization of the context index

The DBMS_XDBT package also contains a set of package variables that describe the configuration settings for the index. These are intended to cover the basic customizations that installations may require, but is by no means a complete set.

The DBMS_XDBT package can be used in the following fashion:

- Customize the package to set up the appropriate configuration.
- Drop any existing index preferences using the dropPreferences() procedure.
- Create new index preferences using the createPreferences() procedure.
- Create the ConText index using the createIndex() procedure.
- Set up automatic synchronization of the index using the configureAutoSync() procedure.

Procedures and Functions of DBMS_XDBT

Table 33–1 Summary of Procedures and Functions of DBMS_XDBT

Procedure/Function	Description
dropPreferences() on page 33-3	Drops any existing preferences.
createPreferences() on page 33-3	Creates preferences required for the ConText index on the XML DB hierarchy.
createDatastorePref() on page 33-3	Creates a USER datastore preference for the ConText index.
createFilterPref() on page 33-4	Creates a filter preference for the ConText index.
createLexerPref() on page 33-4	Creates a lexer preference for the ConText index.
createWordlistPref() on page 33-5	Creates a stoplist for the ConText index.
createStoplistPref() on page 33-5	Creates a section group for the ConText index.
createStoragePref() on page 33-6	Creates a wordlist preference for the ConText index.

Table 33–1 Summary of Procedures and Functions of DBMS_XDBT (Cont.)

Procedure/Function	Description
createSectiongroupPref() on page 33-6	Creates a storage preference for the ConText index.
createIndex() on page 33-6	Creates the ConText index on the XML DB hierarchy.
configureAutoSync() on page 33-7	Configures the ConText index for automatic maintenance (SYNC).

dropPreferences()

Description

This procedure drops any previously created preferences for the `ConText` index on the XML DB hierarchy.

Syntax

```
PROCEDURE dropPreferences;
```

createPreferences()

Description

This procedure creates a set of default preferences based on the configuration settings.

Syntax

```
PROCEDURE createPreferences;
```

createDatastorePref()

Description

This procedure creates a `USER` datastore preference for the ConText index on the XML DB hierarchy.

- The name of the datastore preference can be modified; see the `DatastorePref` configuration setting.

- The default `USER` datastore procedure also filters the incoming document. The `DBMS_XDBT` package provides a set of configuration settings that control the filtering process.
- The `SkipFilter_Types` array contains a list of regular expressions. Documents with a mime type that matches one of these expressions are not indexed. Some of the properties of the document metadata, such as author, remain unindexed.
  - The `NullFilter_Types` array contains a list of regular expressions. Documents with a mime type that matches one of these expressions are not filtered; however, they are still indexed. This is intended to be used for documents that are text-based, such as HTML, XML and plain-text.
  - All other documents use the INSO filter through the IFILTER API.

Syntax

```
PROCEDURE createDatastorePref;
```

createFilterPref()

Description

This procedure creates a `NULL` filter preference for the `ConText` index on the XML DB hierarchy.

- The name of the filter preference can be modified; see `FilterPref` configuration setting.
- The `USER` datastore procedure filters the incoming document; see `createDatastorePref` for more details.

Syntax

```
PROCEDURE createFilterPref;
```

createLexerPref()

Description

This procedure creates a BASIC `lexer` preference for the `ConText` index on the XML DB hierarchy.

- The name of the `lexer` preference can be modified; see `LexerPref` configuration setting. No other configuration settings are provided.

- MultiLexer preferences are not supported.
- Base letter translation is turned on by default.

Syntax

```
PROCEDURE createLexerPref;
```

createWordlistPref()

Description

This procedure creates a wordlist preference for the ConText index on the XML DB hierarchy.

- The name of the wordlist preference can be modified; see the WordlistPref configuration setting. No other configuration settings are provided.
- FUZZY_MATCH and STEMMER attributes are set to AUTO (auto-language detection)

Syntax

```
PROCEDURE createWordlistPref;
```

createStoplistPref()

Description

This procedure creates a stoplist for the ConText index on the XML DB hierarchy.

- The name of the stoplist can be modified; see the StoplistPref configuration setting.
- Numbers are not indexed.
- The StopWords array is a configurable list of stopwords. These are meant to be stopwords in addition to the set of stopwords in CTXSYS.DEFAULT_STOPLIST.

Syntax

```
PROCEDURE createStoplistPref;
```

createStoragePref()

Description

This procedure creates a `BASIC_STORAGE` preference for the `ConText` index on the XML DB hierarchy.

- The name of the storage preference can be modified; see the `StoragePref` configuration setting.
- A tablespace can be specified for the tables and indexes comprising the `ConText` index; see the `IndexTablespace` configuration setting.
- Prefix and Substring indexing are not turned on by default.
- The `I_INDEX_CLAUSE` uses key compression.

Syntax

```
PROCEDURE createStoragePref;
```

createSectiongroupPref()

Description

This procedure creates a section group for the `ConText` index on the XML DB hierarchy.

- The name of the section group can be changed; see the `SectiongroupPref` configuration setting.
- The HTML sectioner is used by default. No zone sections are created by default. If the vast majority of documents are XML, consider using the `AUTO_SECTION_GROUP` or the `PATH_SECTION_GROUP`; see the `SectionGroup` configuration setting.

Syntax

```
PROCEDURE createSectiongroupPref;
```

createIndex()

Description

This procedure creates the `ConText` index on the XML DB hierarchy.

Syntax

```
PROCEDURE createIndex;
```

Notes

- The name of the index can be changed; see the `IndexName` configuration setting.
- Set the `LogFile` configuration parameter to enable `ROWID` logging during index creation.
- Set the `IndexMemory` configuration parameter to determine the amount of memory that index creation, and later SYNCs, will use.

configureAutoSync()

Description

This procedure sets up jobs for automatic SYNCs of the `ConText` index.

- The system must be configured for job queues for automatic synchronization. The jobs can be viewed using the `USER_JOBS` catalog views
- The configuration parameter AutoSyncPolicy can be set to choose an appropriate synchronization policy.

The synchronization can be based on one of the following:

SYNC_BY_PENDING_COUNT	The SYNC is triggered when the number of documents in the pending queue is greater than a threshold (See the MaxPendingCount configuration setting) The pending queue is polled at regular intervals (See the CheckPendingCountInterval configuration parameter) to determine if the number of documents exceeds the threshold
SYNC_BY_TIME	The SYNC is triggered at regular intervals. See the SyncInterval configuration parameter.
SYNC_BY_PENDING_COUNT_AND_TIME	A combination of both SYNC_BY_PENDING_COUNT and SYNC_BY_TIME

Syntax

```
PROCEDURE configureAutoSync;
```

Customizing the DBMS_XDBT package

The DBMS_XDBT package can be customized in one of two ways.

- Using a PL/SQL procedure, or an anonymous block, to set the relevant package variables (configuration settings), and then executing the procedures in this package.
- The more general approach is to introduce the appropriate customizations by modifying this package in place, or as a copy.

Please note that the system must be configured to use job queues, and that the jobs can be viewed through the USER_JOBS catalog views.

This section describes the configuration settings, or package variables, available to customize the DBMS_XDBT package.

General Indexing Settings

The following table lists configuration settings that are relevant to general indexing.

Parameter	Default Values	Description
IndexName	XDB$CI	The name of the `ConText` index.
IndexTablespace	XDB$RESINFO	The tablespace used by tables and indexes comprising the `ConText` index.
IndexMemory	128M	Memory used by index creation and Sync. This must be less than (or equal to) the `MAX_INDEX_MEMORY` system parameter. The `MAX_INDEX_MEMORY` system parameter (see the `CTX_ADMIN` package) must be greater than or equal to the `IndexMemory` setting.
LogFile	'XdbCtxLog'	The logfile used for `ROWID` logging during index creation. The `LOG_DIRECTORY` system parameter must be set already. Set this to `NULL` to turn off `ROWID` logging. The `LOG_DIRECTORY` system parameter (see the `CTX_ADMIN` package) must be set to enable `ROWID` logging.

Filtering Settings

The following table lists configuration settings that control the filtering of documents in the XML DB hierarchy.

Parameter	Default Value(s)	Description
SkipFilter_Types	image/%, audio/%, video/%, model/%	List of mime types that should not be indexed.
NullFilter_Types	text/plain, text/html, text/xml	List of mime types that do not need to use the INSO filter. Use this for text-based documents.
FilterPref	XDB$CI_FILTER	Name of the filter preference.

Sectioning and Section Group Settings

The following table lists configuration settings relevant to the sectioner.

Parameter	Default Value	Description
SectionGroup	HTML_SECTION_GROUP	Default sectioner to use. Consider using `PATH_SECTION_GROUP` or `AUTO_SECTION_GROUP` if the repository contains mainly XML documents.
SectiongroupPref	XDB$CI_SECTIONGROUP	Name of the section group.

Stoplist Settings

The following table lists stoplist configuration settings.

Parameter	Default Value(s)	Description
StoplistPref	XDB$CI_STOPLIST	Name of the stoplist.
StopWords	0..9 'a'..'z' 'A'..'Z'	List of stopwords, in addition to stopwords specified in `CTXSYS.DEFAULT_STOPLIST`.

Other Preference Settings

The following table lists settings for other index preferences.

Parameter	Default Value	Description
DatastorePref	XDB$CI_DATASTORE	The name of the datastore preference.

Parameter	Default Value	Description
StoragePref	XDB$CI_STORAGE	The name of the storage preference.
WordlistPref	XDB$CI_WORDLIST	The name of the wordlist preference.
DefaultLexerPref	XDB$CI_DEFAULT_ LEXER	The name of the default lexer preference.

Index SYNC settings

The following table lists settings that control when and how the `ConText` index is synchronized.

Parameter	Default Value(s)	Description
AutoSyncPolicy	SYNC_BY_PENDING_COUNT	Indicates when the index should be SYNCed. Can be one of: - `SYNC_BY_PENDING_COUNT,` - `SYNC_BY_TIME,` or - `SYNC_BY_PENDING_COUNT_AND_TIME.`
MaxPendingCount	2	Maximum number of documents in the `CTX_USER_PENDING` queue for this index before an index SYNC is triggered. Applies only if the `AutoSyncPolicy` is one of: - `SYNC_BY_PENDING_COUNT`, or - `SYNC_BY_PENDING_COUNT_AND_TIME.`
CheckPendingCountInterval	10 minutes	Indicates how often, in minutes, the pending queue should be checked. Applies only if the `AutoSyncPolicy` is one of: - `SYNC_BY_PENDING_COUNT,` or - `SYNC_BY_PENDING_COUNT_AND_TIME.`
SyncInterval	60 minutes	Indicates how often, in minutes, the index should be SYNCed. Applies only if the AutoSyncPolicy is one of: - `SYNC_BY_TIME`, or - `SYNC_BY_PENDING_COUNT_AND_TIME`

Part VII

XML Database Support: SQLX

This section contains the following chapters:

34

SQLX Functions and Operators

This chapter contains the following sections:

- SQLX Package

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*
- *Oracle9i SQL Reference*

SQLX Package

Description of SQLX

The SQLX functions supported with this release follow the SQLX standard.

Functions and Operators of SQLX

Table 34–1 Summary of Functions and Operators of SQLX

Function/Operator	Description
XMLElement() on page 34-2	Creates an XML Element.
XMLForest() on page 34-4	Creates an XML Fragment from passed-in components.
XMLColAttVal() on page 34-4	Creates an XML fragment and then expands the resulting XML so that each XML fragment has the name "column" with the attribute "name"
ExtractValue() on page 34-5	Takes as arguments an `XMLType` instance and an XPath expression and returns a scalar value of the resultant node.
XMLTransform() on page 34-5	Takes as arguments an `XMLType` instance and an XSL style sheet, which is itself a form of `XMLType` instance. It applies the style sheet to the instance and returns an `XMLType`.
XMLSequence() on page 34-6	Takes input and returns either a varray of the top-level nodes in the `XMLType`, or an `XMLSequence` type an XML document for each row of the cursor.
XMLConcat() on page 34-7	Takes as input a series of `XMLType` instances, concatenates the series of elements for each row, and returns the concatenated series.
UpdateXML() on page 34-7	Takes as arguments an `XMLType` instance and an XPath-value pair, and returns an `XMLType` instance with the updated value.

XMLElement()

Description

`XMLElement` takes an element name for *`identifier`*, an optional collection of attributes for the element, and arguments that make up the element's content. It returns an instance of type `XMLType`. `XMLElement` is similar to `SYS_XMLGen` except that `XMLElement` can include attributes in the XML returned, but it does not accept formatting using the `XMLFormat` object.

The `XMLElement` function is typically nested to produce an XML document with a nested structure, as in the example in the following section.

You must specify a value for *`identifier`*, which Oracle uses as the enclosing tag. The identifier does not have to be a column name or column reference. It cannot be an expression or null.

In the *`XML_attributes_clause`*, if the *`value_expr`* is null, then no attribute is created for that value expression. The type of *`value_expr`* cannot be an object type or collection.

The objects that make up the element content follow the `XMLATTRIBUTES` keyword.

- If *`value_expr`* is a scalar expression, then you can omit the AS clause, and Oracle uses the column name as the element name.
- If *`value_expr`* is an object type or collection, then the AS clause is mandatory, and Oracle uses the specified *`c_alias`* as the enclosing tag.
- If *`value_expr`* is null, then no element is created for that value expression.

Syntax

```
XMLELEMENT(   elementname IN VARCHAR2,
              [XMLATTRIBUTES(),]
               value_expr,
               ...)
               RETURN XMLType

XMLATTRIBUTES( value_expr [AS identifier],
               ...)
               RETURN XMLType
```

Parameter	Description
elementname	Indicates the name of the XML element to be generated
value_expr	This must be a scalar value for XMLATTRIBUTES. It can be a nested XMLELEMENT call for XMLELEMENT.

XMLForest()

Description

`XMLForest` converts each of its argument parameters to XML, and then returns an XML fragment that is the concatenation of these converted arguments.

- If *`value_expr`* is a scalar expression, then you can omit the AS clause, and Oracle uses the column name as the element name.
- If *`value_expr`* is an object type or collection, then the AS clause is mandatory, and Oracle uses the specified *`c_alias`* as the enclosing tag.
- If *`value_expr`* is null, then no element is created for that *`value_expr`*.

Syntax

```
XMLFOREST( value_expr [AS identifier],
          ...)
          RETURN XMLType
```

Parameter	Description
value_expr	May be a scalar, object, or nested call.

XMLColAttVal()

Description

`XMLColAttVal` creates an XML fragment and then expands the resulting XML so that each XML fragment has the name "column" with the attribute "name". You can use the AS *`c_alias`* clause to change the value of the `name` attribute to something other than the column name.

You must specify a value for *`value_expr`*. If *`value_expr`* is null, then no element is returned.

Restriction: You cannot specify an object type column for *`value_expr`*.

Syntax

```
XMLCOLATTVAL( value_expr [AS identifier],
             ...)
             RETURN XMLType
```

Parameter	Description
value_expr	May be a scalar, object, or nested call.

ExtractValue()

Description

The EXTRACTVALUE function takes as arguments an XMLType instance and an XPath expression and returns a scalar value of the resultant node. The result must be a single node and be either a text node, attribute, or element. If the result is an element, the element must have a single text node as its child, and it is this value that the function returns. If the specified XPath points to a node with more than one child, or if the node pointed to has a non-text node child, Oracle returns an error.

For documents based on XML schemas, if Oracle can infer the type of the return value, then a scalar value of the appropriate type is returned. Otherwise, the result is of type VARCHAR2. For documents that are not based on XML schemas, the return type is always VARCHAR2.

Syntax

```
EXTRACTVALUE( instance   IN  XMLType,
              xpath      IN  VARCHAR2,
             [namespace  IN  VARCHAR2])
             RETURN value
```

Parameter	Description
instance	The XMLType to extract the data from.
xpath	XPath expression as a string
namespace	Optional argument specifying namespace declarations.

XMLTransform()

Description

XMLTransform takes as arguments an XMLType instance and an XSL style sheet, which is itself a form of XMLType instance. It applies the style sheet to the instance and returns an XMLType.

This function is useful for organizing data according to a style sheet as you are retrieving it from the database.

Syntax

```
XMLTRANSFORM( instance    IN   XMLType,
              xslt        IN   XMLType)
              RETURN XMLType
```

Parameter	Description
instance	The XMLType instance to be transformed.
xslt	The XSLT document to transform the instance.

XMLSequence()

Description

`XMLSequence` has two forms:

- The first form takes as input an `XMLType` instance and returns a varray of the top-level nodes in the `XMLType`.
- The second form takes as input a `REFCURSOR` instance, with an optional instance of the `XMLFormat` object, and returns as an `XMLSequence` type an XML document for each row of the cursor.

Because `XMLSequence` returns a collection of `XMLType`, you can use this function in a `TABLE` clause to unnest the collection values into multiple rows, which can in turn be further processed in the SQL query.

Syntax

```
XMLSEQUENCE( { input IN XMLType | input IN SYS_REFCURSOR,
               [format IN XMLFormat] })
             RETURN XMLType
```

Parameter	Description
input	The data to be split up.
format	The optional XMLFormat to be used to format the output from the cursor.

XMLConcat()

Description

`XMLConcat` takes as input a series of `XMLType` instances, concatenates the series of elements for each row, and returns the concatenated series. `XMLConcat` is the inverse of `XMLSequence`.

Null expressions are dropped from the result. If all the value expressions are null, then the function returns null.

The options are described in the following table.

Syntax	Description
XMLCONCAT(data IN XMLSEQUENCETYPE) RETURN XMLType	Takes in one argument of type XMLSequenceType, and retruns an XMLType which is the concatenation of all XMLTypes in the sequence.
XMLCONCAT (arg1 IN XMLType, ...) RETURN XMLType	This is an n-ary function that constructs a forest of XML elements from its arguments. If an argument is the `NULL` value, then it is quietly dropped from the result. If all of the arguments are the `NULL` value, then the `NULL` value is returned.

Parameter	Description
data	The XMLSequenceType to be concatenated together.
arg1, arg2, ...	The XMLType arguments to be concatenated together.

UpdateXML()

`UPDATEXML` takes as arguments an `XMLType` instance and an XPath-value pair, and returns an `XMLType` instance with the updated value. If *`XPath_string`* is an XML element, then the corresponding *`value_expr`* must be an `XMLType` instance. If *`XPath_string`* is an attribute or text node, then the *`value_expr`* can be any scalar datatype. The datatypes of the target of *`XPath_string`* and the *`value_expr`* must match.

If you update an XML element to null, Oracle removes the attributes and children of the element, and the element becomes empty. If you update the text node of an element to null, Oracle removes the text value of the element, and the element itself remains but is empty.

In most cases, this function materializes an XML document in memory and updates the value. However, `UPDATEXML` is optimized for `UPDATE` statements on object-relational columns so that the function updates the value directly in the column. This optimization requires the following conditions:

- The *`XMLType_instance`* must be the same as the column in the `UPDATE` ... `SET` clause.
- The *`XPath_string`* must resolve to scalar content.

Syntax

```
UPDATEXML( instance  IN XMLType,
          [xpath     IN VARCHAR2,
          value_expr)], ...,
          [namespace IN VARCHAR2] )
         RETURN XMLType
```

Parameter	Description
instance	The XMLType to be updated.
xpath	The location in the XMLType to be changed.
value_expr	The value to update the location to. May be an XMLType or a string, depending on the location pointed to.
namespace	Optional argument specifying namespace declarations.

Part VIII

XML Database Support: Database URI Types

This section contains the following chapter:

35

Database URI Types

This chapter contains the following sections:

See Also:

- *Oracle9i XML Database Developer's Guide - Oracle XML DB*

URI Support

Description

Oracle9*i* supports the `UriType` family of types that can be used to store and query Uri-refs inside the database. The `UriType` itself is an abstract object type and the `HttpUriType, XDBUriType` and `DBUriType` are subtypes of it.

You can create a `UriType` column and store instances of the `DBUriType, XDBUriType` or the `HttpUriType` inside of it.

You can also define your own subtypes of the `UriType` to handle different URL protocols.

Oracle9*i* also provides a `UriFactory` package that can be used as a factory method to automatically generate various instances of these `UriTypes` by scanning the prefix (for example, `http://` or `/oradb`). You can also register your subtype and provide the prefix that you support. For instance, if you have written a subtype to handle the gopher protocol, you can register the prefix `gopher://` to be handled by your subtype. The `UriFactory` will then generate your subtype instance for any URL starting with that prefix.

Table 35–1 URI Support Types

Type	Description
UriType Super Type on page 35-3	An abstract type that is the base type of the database URI types.
HttpUriType Subtype on page 35-7	The HTTP implementation of URI type.
DBUriType Subtype on page 35-12	The implementation of URI type referencing database objects.
XDBUriType Subtype on page 35-18	The implementation of URI type referencing Oracle XML DB objects.
UriFactory Package on page 35-22	Package that generates the appropriate URI type for the given URI.

UriType Super Type

Description of UriType

The `UriType` is the abstract super type. It provides a standard set of functions to get the value pointed to by the URI. The actual implementation of the protocol must be defined by the subtypes of this type.

Instances of this type cannot be created directly. However, you can create columns of this type and store subtype instances in it.

For example,

```
    create table uri_tab ( url uritype);

insert into uri_tab values
  (httpuritype.createuri('http://www.oracle.com'));
insert into uri_tab values
  (dburitype.createuri('/SCOTT/EMPLOYEE/ROW[ENAME="Jack"]'));
```

Now you can select from the column without having to know what instance of the URL is actually stored. The following line of code would retrieve both the HTTP URL and the DBUri-ref:

```
select e.url.getclob() from uri_tab e;
```

Functions and Procedures of UriType

Table 35–2 Summary of Functions and Procedures of UriType

Function	Description
getBlob() on page 35-4	Returns the BLOB located at the address specified by the URL.
getClob() on page 35-4	Returns the CLOB located at the address specified by the URL.
getContentType() on page 35-5	Returns the URL, in escaped format, stored inside the `UriType` instance.
getExternalUrl() on page 35-5	Returns the URL, in escaped format, stored inside the `UriType` instance.
getUrl() on page 35-5	Returns the URL, in non-escaped format, stored inside the `UriType` instance.
getXML() on page 35-6	Returns the XMLType located at the address specified by the URL.

getBlob()

Description

This function returns the BLOB located at the address specified by the URL. This function can be overridden in the subtype instances. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getBlob() RETURN blob;	This function returns the BLOB located at the address specified by the URL.
MEMBER FUNCTION getBlob(content OUT VARCHAR2) RETURN blob;	This function returns the BLOB located at the address specified by the URL and and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

getClob()

Description

This function returns the CLOB located at the address specified by the URL. This function can be overridden in the subtype instances. This function returns either a permanent CLOB or a temporary CLOB. If a temporary CLOB is returned, it must be freed. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getClob() RETURN clob;	This function returns the CLOB located at the address specified by the URL.
MEMBER FUNCTION getClob(content OUT VARCHAR2) RETURN clob;	This function returns the CLOB located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

getContentType()

Description

This function returns the content type of the document pointed to by the URI. This function can be overridden in the subtype instances. This function returns the content type as VARCHAR2.

Syntax

```
MEMBER FUNCTION getContentType() RETURN VARCHAR2;
```

getExternalUrl()

Description

This function returns the URL, in escaped format, stored inside the `UriType` instance. The subtype instances override this member function to provide additional semantics. For instance, the `HttpUriType` function does not store the prefix `http://` in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the `getExternalUrl` function or the `getUrl` function to get to the URL value instead of using the attribute present in the `UriType` instance.

Syntax

```
MEMBER FUNCTION getExternalUrl() RETURN varchar2;
```

getUrl()

Description

This function returns the URL, in non-escaped format, stored inside the `UriType` instance. The subtype instances override this member function to provide additional semantics. For instance, the `HttpUriType` function does not store the prefix `http://` in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the `getExternalUrl` function or the `getUrl` function to get to the URL value instead of using the attribute present in the `UriType` instance.

Syntax

```
MEMBER FUNCTION getUrl() RETURN varchar2;
```

getXML()

Description

This function returns the XMLType located at the address specified by the URL. This function can be overridden in the subtype instances. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getXML() RETURN XMLType;	This function returns the XMLType located at the address specified by the URL.
MEMBER FUNCTION getXML(content OUT VARCHAR2) RETURN XMLType;	This function returns the XMLType located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

HttpUriType Subtype

Description of HttpUriType

The `HttpUriType` is a subtype of the `UriType` that provides support for the HTTP protocol. This uses the `UTL_HTTP` package underneath to access the HTTP URLs. Proxy and secure wallets are not supported in this release.

The following example creates a uri table to store the Http instances

```
create table uri_tab ( url httpuritype);
```

Insert the Http instance.

```
insert into uri_tab values
    (httpuritype.createUri('http://www.oracle.com'));
```

Generate the HTML

```
select e.url.getclob() from uri_tab e;
```

Functions and Procedures of HttpUriType

Table 35–3 Summary of Functions and Procedures of HttpUriType

Function	Description
createUri() on page 35-8	Creates an instance of HTTPURIType from the given URI.
getBlob() on page 35-8	Returns the BLOB located at the address specified by the URL.
getClob() on page 35-9	Returns the CLOB located at the address specified by the URL.
getContentType() on page 35-9	Returns the content type of the document pointed to by the URI.
getExternalUrl() on page 35-9	Returns the URL, in escaped format, stored inside the `UriType` instance.
getUrl() on page 35-10	Returns the URL, in non-escaped format, stored inside the `UriType` instance.
getXML() on page 35-10	Returns the XMLType located at the address specified by the URL
httpUriType() on page 35-10	Creates an instance of HTTPURIType from the given URI.

createUri()

Description

This static function constructs a `HttpUriType` instance. The `HttpUriType` instance does not contain the prefix `http://` in the stored URL.

Syntax

```
STATIC FUNCTION createUri(url IN varchar2) RETURN HttpUriType;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string containing a valid HTTP URL. The URL string is expected in escaped format. For example, non-url characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.

getBlob()

Description

This function returns the BLOB located at the address specified by the HTTP URL. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getBlob() RETURN blob;	This function returns the BLOB located at the address specified by the HTTP URL.
MEMBER FUNCTION getBlob(content OUT VARCHAR2) RETURN blob;	This function returns the BLOB located at the address specified by the HTTP URL and and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

getClob()

Description

This function returns the CLOB located at the address specified by the HTTP URL. This function returns either a permanent CLOB or a temporary CLOB. If a temporary CLOB is returned, it must be freed. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getClob() RETURN clob;	Returns the CLOB located at the address specified by the HTTP URL.
MEMBER FUNCTION getClob(content OUT VARCHAR2) RETURN clob;	Returns the CLOB located at the address specified by the HTTP URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

getContentType()

Description

Returns the content type of the document pointed to by the URI.

Syntax

```
MEMBER FUNCTION getContentType() RETURN VARCHAR2;
```

getExternalUrl()

Description

This function returns the URL, in escaped format, stored inside the `HttpUriType` instance. The subtype instances override this member function to provide additional semantics. The `HttpUriType` function does not store the prefix `http://` in the URL itself. When generating the external URL, it prepends the prefix and generates it.

Syntax

```
MEMBER FUNCTION getExternalUrl() RETURN varchar2;
```

getUrl()

Description

This function returns the URL, in non-escaped format, stored inside the `HttpUriType` instance.

Syntax

```
MEMBER FUNCTION getUrl() RETURN varchar2;
```

getXML()

Description

This function returns the XMLType located at the address specified by the URL. An error tis thrown if the address does not point to a valid XML doc. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getXML() RETURN XMLType;	This function returns the XMLType located at the address specified by the URL.
MEMBER FUNCTION getXML(content OUT VARCHAR2) RETURN XMLType;	This function returns the XMLType located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

httpUriType()

Description

This constructs a `HttpUriType` instance. The `HttpUriType` instance does not contain the prefix `http://` in the stored URL.

Syntax

```
CONSTRUCTOR FUNCTION httpUriType(url IN varchar2);
```

Parameter	IN / OUT	Description
url	(IN)	The URL string containing a valid HTTP URL. The URL string is expected in escaped format. For example, non-url characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.

DBUriType Subtype

Description of DBUriType

The DBUriType is a subtype of the UriType that provides support for of DBUri-refs. A DBUri-ref is an intra-database URL that can be used to reference any row or row-column data in the database. The URL is specified as an XPath expression over a XML visualization of the database. The schemas become elements which contain tables and views. These tables and view further contain the rows and columns inside them.

For example, the virtual document that a user scott can see can be:

```
<?xml version="1.0"?>
<ORADB>
  <SCOTT>
    <EMPLOYEE>
      <ROWSET>
         <ROW>
          <EMPNO>100</EMPNO>
          <ENAME>John</ENAME>
          <ADDRESS>
             <STREET>100 Main Street</STREET>
             <CITY>Jacksonville</CITY>
             <STATE>FL</STATE>
             <ZIP>32607</ZIP>
           </ADDRESS>
         </ROW>
         <ROW>
           <EMPNO>200</EMPNO>
           <ENAME>Jack</ENAME>
           <ADDRESS>
             <STREET>200 Front Street</STREET>
             <CITY>San Francisco</CITY>
             <STATE>CA</STATE>
             <ZIP>94011</ZIP>
           </ADDRESS>
         </ROW>
       </ROWSET>
     </EMPLOYEE>
   </SCOTT>
 </ORADB>
```

Hence, to reference the State attribute inside the employee table, you can formulate a DBUri-ref as shown:

```
/ORADB/SCOTT/EMPLOYEE/ROW[ENAME="Jack"]/ADDRESS/STATE
```

You can use the DBUriType to create instances of these and store them in columns.

Create a table

```
create table dburi_tab (dburl dburitype);
```

Insert values

```
insert into dburi_tab values (
   dburitype.createUri(
        '/ORADB/SCOTT/EMPLOYEE/ROW[ENAME="Jack"]/ADDRESS/STATE'));
select e.dburl.getclob() from dburi_tab e;
```

Will return

```
<?xml version="1.0"?>
<STATE>CA</STATE>
```

You can also generate the DBUri-ref dynamically using the `SYS_DBURIGEN` SQL function.

For example you can generate a DBUri-ref to the state attribute as shown:

```
select sys_dburigen( e.ename, e.address.state) AS urlcol
     from scott.employee e;
```

Functions and Procedures of DBUriType

Table 35–4 Summary of Functions and Procedures of DBUriType

Function	Description
createUri() on page 35-14	Constructs a `DBUriType` instance.
DBUriType() on page 35-14	Creates an instance of DBURIType from the given URI.
getBlob() on page 35-15	Returns the BLOB located at the address specified by the `DBUriType` instance.
getClob() on page 35-15	Returns the CLOB located at the address specified by the `DBUriType` instance.

Table 35–4 Summary of Functions and Procedures of DBUriType (Cont.)

Function	Description
getContentType() on page 35-16	Returns the content type of the document pointed to by the URI.
getExternalUrl() on page 35-16	Returns the URL, in escaped format, stored inside the `DBUriType` instance.
getUrl() on page 35-16	Returns the URL, in non-escaped format, stored inside the `DBUriType` instance.
getXML() on page 35-17	Returns the XMLType located at the address specified by the URL

createUri()

Description

This static function constructs a `DBUriType` instance. Parses the URL given and creates a `DBUriType` instance.

Syntax

```
STATIC FUNCTION createUri( url IN varchar2) RETURN DBUriType;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string, in escaped format, containing a valid DBUri reference.

DBUriType()

Description

This constructs a `DBUriType` instance.

Syntax

```
CONSTRUCTOR FUNCTION DBUriType( url IN varchar2);
```

Parameter	IN / OUT	Description
url	(IN)	The URL string containing a valid DBUri reference.The URL string is expected in escaped format. For example, non-URL characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.

getBlob()

Description

This function returns the BLOB located at the address specified by the URL. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getBlob() RETURN blob;	This function returns the BLOB located at the address specified by the URL.
MEMBER FUNCTION getBlob(content OUT VARCHAR2) RETURN blob;	This function returns the BLOB located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

getClob()

Description

This function returns the CLOB located at the address specified by the `DBUriType` instance. If a temporary CLOB is returned, it must be freed. The document returned may be an XML document or a text document. When the DBUri-ref identifies an element in the XPath, the result is a well-formed XML document. On the other hand, if it identifies a text node (using the `text()` function), then what is returned is only the text content of the column or attribute. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getClob() RETURN clob;	Returns the CLOB located at the address specified by the DBUirType instance.
MEMBER FUNCTION getClob(content OUT VARCHAR2) RETURN clob;	Returns the CLOB located at the address specified by the DBUirType instance and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

getContentType()

Description

This function returns the content type of the document pointed to by the URI.

Syntax

```
MEMBER FUNCTION getContentType() RETURN VARCHAR2;
```

getExternalUrl()

Description

This function returns the URL, in escaped format, stored inside the `DBUriType` instance. The `DBUri` servlet URL that processes the DBUriType has to be appended before using the escaped URL in web pages.

Syntax

```
MEMBER FUNCTION getExternalUrl() RETURN varchar2;
```

getUrl()

Description

This function returns the URL, in non-escaped format, stored inside the DBUriType instance.

Syntax

```
MEMBER FUNCTION getUrl() RETURN varchar2;
```

getXML()

Description

This function returns the XMLType located at the address specified by the URL. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getXML() RETURN XMLType;	This function returns the XMLType located at the address specified by the URL.
MEMBER FUNCTION getXML(content OUT VARCHAR2) RETURN XMLType;	This function returns the XMLType located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

XDBUriType Subtype

Description of XDBUriType

XDBUriType is a new subtype of URIType. It provides a way to expose documents in the Oracle XML DB hierarchy as URIs that can be embedded in any URIType column in a table. The URL part of the URI is the hierarchical name of the XML document it refers to. The optional fragment part uses the XPath syntax, and is separated from the URL part by '#'. The more general XPointer syntax for specifying a fragment is not currently supported.

Functions and Procedures of XDBUriType

Table 35–5 Summary of Functions and Procedures XDBUriType Functions

Function	Description
createUri() on page 35-18	Returns the UriType corresponding to the specified URL.
getBlob() on page 35-19	Returns the BLOB corresponding to the contents of the document specified by the XDBUriType instance.
getClob() on page 35-15	Returns the CLOB corresponding to the contents of the document specified by the `XDBUriType` instance.
getContentType() on page 35-20	Returns the content type of the document pointed to by the URI.
getExternalUrl() on page 35-16	Returns the URL, in escaped format, stored inside the `XDBUriType` instance.
getUrl() on page 35-16	Returns the URL, in non-escaped format, stored inside the `XDBUriType` instance.
getXML() on page 35-21	Returns the XMLType corresponding to the contents of the document specified by the URL.
XDBUriType() on page 35-21	Creates an instance of XDBURIType from the given URI.

createUri()

Description

This static function constructs a `XDBUriType` instance. Parses the URL given and creates a `XDBUriType` instance.

Syntax

```
STATIC FUNCTION createUri(url IN varchar2) RETURN XDBUriType;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string, in escaped format, containing a valid XDBUri reference.

getBlob()

Description

This function returns the BLOB located at the address specified by the XDBUriType instance. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getBlob() RETURN blob;	This function returns the BLOB located at the address specified by the URL.
MEMBER FUNCTION getBlob(content OUT VARCHAR2) RETURN blob;	This function returns the BLOB located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

getClob()

Description

This function returns the CLOB located at the address specified by the `XDBUriType` instance. If a temporary CLOB is returned, it must be freed. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getClob() RETURN clob;	Returns the CLOB located at the address specified by the DBUirType instance.

Syntax	Description
MEMBER FUNCTION getClob(content OUT VARCHAR2) RETURN clob;	Returns the CLOB located at the address specified by the DBUirType instance and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

getContentType()

Description

This function returns the content type of the document pointed to by the URI. This function returns the content type as VARCHAR2.

Syntax

```
MEMBER FUNCTION getContentType() RETURN VARCHAR2;
```

getExternalUrl()

Description

This function returns the URL, in escaped format, stored inside the `XDBUriType` instance.

Syntax

```
MEMBER FUNCTION getExternalUrl() RETURN varchar2;
```

getUrl()

Description

This function returns the URL, in non-escaped format, stored inside the `XDBUriType` instance.

Syntax

```
MEMBER FUNCTION getUrl() RETURN varchar2;
```

getXML()

Description

This function returns the XMLType located at the address specified by the URL. The options are described in the following table.

Syntax	Description
MEMBER FUNCTION getXML() RETURN XMLType;	This function returns the XMLType located at the address specified by the URL.
MEMBER FUNCTION getXML(content OUT VARCHAR2) RETURN XMLType;	This function returns the XMLType located at the address specified by the URL and the content type.

Parameter	IN / OUT	Description
content	(OUT)	Content type of the document to which URI is pointing.

XDBUriType()

Description

This constructs a `XDBUriType` instance.

Syntax

```
CONSTRUCTOR FUNCTION XDBUriType( url IN varchar2);
```

Parameter	IN / OUT	Description
url	(IN)	The URL string containing a valid XDBUri reference.The URL string is expected in escaped format. For example, non-URL characters are represented as the hexadecimal value for the UTF-8 encoding of those characters.

UriFactory Package

Description of UriFactory

The `UriFactory` package contains factory methods that can be used to generate the appropriate instance of the URI types without having to hard code the implementation in the program.

The `UriFactory` package also provides the ability to register new subtypes of the `UriType` to handle various other protocols not supported by Oracle9*i*. For example, one can invent a new protocol `ecom://` and define a subtype of the `UriType` to handle that protocol and register it with `UriFactory`. After that any factory method would generate the new subtype instance if it sees the `ecom://` prefix.

For example,

```
create table url_tab (urlcol varchar2(20));
```

Insert a Http reference

```
insert into url_tab values ('http://www.oracle.com');
```

Insert a DBuri-ref reference

```
insert into url_tab values ('/SCOTT/EMPLOYEE/ROW[ENAME="Jack"]');
```

Create a new type to handle a new protocol called `ecom://`

```
create type EComUriType under UriType
(
  overriding member function getClob() return clob,
  overriding member function getBlob() return blob,
     -- not supported
  overriding member function getExternalUrl() return varchar2,
  overriding member function getUrl() return varchar2,

  -- MUST NEED THIS for registering with the url handler
  static member function createUri(url in varchar2)
         return EcomUriType
);
```

Register a new protocol handler:

```
begin
     -- register a new handler for ecom:// prefixes. The handler
     -- type name is ECOMURITYPE, schema is SCOTT
     -- Ignore the prefix case, when comparing and also strip
     -- the prefix before calling the createUri function
     urifactory.registerHandler('ecom://','SCOTT','
                                        ECOMURITYPE', true,true);
end;

insert into url_tab values ('ECOM://company1/company2=22/comp');
```

Now use the factory to generate the instances

```
select urifactory.getUri(urlcol) from url_tab;
```

Would now generate

```
HttpUriType('www.oracle.com');
        -- an Http uri type instance

DBUriType('/SCOTT/EMPLOYEE/ROW[ENAME="Jack"],null);
        -- a DBUriType

EComUriType('company1/company2=22/comp');
        -- a EComUriType instance
```

Functions and Procedures of URIFactory

Table 35–6 Summary of Functions and Procedures of UriFactory

Method	**Description**
getUri() on page 35-24	Returns the correct URL handler for the given URL string.
escapeUri() on page 35-24	Returns a URL in escaped format.
unescapeUri() on page 35-25	Returns a URL in unescaped format.
registerUrlHandler() on page 35-25	Registers a particular type name for handling a particular URL.
unRegisterUrlHandler() on page 35-26	Unregisters a URL handler.

getUri()

Description

This factory method returns the correct URL handler for the given URL string. It returns a subtype instance of the `UriType` that can handle the protocol. By default, it always creates an `XDBUriType` instance, if it cannot resolve the URL. A URL handler can be registered for a particular prefix using the `registerUrlHandler()` function. If the prefix matches, `getUrl()` would then use that subtype.

Syntax

```
FUNCTION getUrl(url IN Varchar2) RETURN UriType;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string, in escaped format, containing a valid HTTP URL.

escapeUri()

Description

This function returns a URL in escaped format. The subtype instances override this member function to provide additional semantics. For instance, the `HttpUriType` does not store the prefix `http://` in the URL itself. When generating the external URL, it appends the prefix and generates it. For this reason, use the `getExternalUrl` function or the `getUrl` function to get to the URL value instead of using the attribute present in the `UriType`.

Syntax

```
MEMBER FUNCTION escapeUri() RETURN varchar2;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string to be returned in escaped format.

unescapeUri()

Description

This function returns a URL in unescaped format. This function is the reverse of the `escapeUri` function. This function scans the string and converts any non-URL hexadecimal characters into the equivalent UTF-8 characters. Since the return type is a VARCHAR2, the characters would be converted into the equivalent characters as defined by the database character set.

Syntax

```
FUNCTION unescapeUri() RETURN varchar2;
```

Parameter	IN / OUT	Description
url	(IN)	The URL string to be returned in unescaped format.

registerUrlHandler()

Description

This procedure registers a particular type name for handling a particular URL. The type specified must be valid and must be a subtype of the `UriType` or one of its subtypes. It must also implement the following static member function:

```
STATIC FUNCTION createUri(url IN varchar2) RETURN <typename>;
```

This function is called by the `getUrl()` function to generate an instance of the type. The `stripprefix` parameter indicates that the prefix must be stripped off before calling this function.

Syntax

```
PROCEDURE registerUrlHandler( prefix IN varchar2,
                              schemaName IN varchar2,
                              typename IN varchar2,
                              ignoreCase IN boolean := true,
                              stripprefix IN boolean := true);
```

Parameter	IN / OUT	Description
prefix	(IN)	The prefix to handle; for example, `http://`.
schemaName	(IN)	The name of the schema where the type resides; case sensitive.
typename	(IN)	The name of the type to handle the URL; case sensitive.
ignoreCase	(IN)	Ignore case when matching prefixes.
stripprefix	(IN)	Strip prefix before generating the instance of the type.

unRegisterUrlHandler()

Description

This procedure unregisters a URL handler. This only unregisters user registered handler prefixes and not predefined system prefixes such as `http://`.

Syntax

```
PROCEDURE unregisterUrlHandler(prefix IN varchar2);
```

Parameter	IN / OUT	Description
prefix	(IN)	The prefix to be unregistered.

Index