

Oracle®

Application Developer's Guide - XML

10g (9.0.4)

Part No. B12099-01

September 2003

ORACLE™

Oracle Application Developer's Guide - XML, 10g (9.0.4)

Part No. B12099-01

Copyright © 2001, 2003 Oracle Corporation. All rights reserved.

Primary Author: Shelley Higgins

Contributing Authors: Omar Alonso, Sandeepan Banerjee, Neerja Bhaat, Kishore Bhamidipati, Stefan Buchta, Raghunath Chintalapati, Robert Dell'immagine, Brajesh Goyal, Robert Hall, Karun K, Stefan Kiritzo, Vishu Krishnamurthy, Murali Krishnaprasad, Olivier LeDiouris, Janet Lee, Wesley Lin, Bryn Llewellyn, Colin McGregor, Ian Macky, Anjana Manian, Becca Martin, Shailendra Mishra, Steve Muench, Bhagat Nainani, Paul Narth, Visar Nimani, Paul Nock, Ami Parekh, Rajesh Raheja, Carol Roston, Frank Rovitto, Tomas Saulys, Mark Scardina, Flora Sun, Prabhu Thukkaram, Rodney Ward, Philipp Weckerle, Manh-Kiet (Allen) Yap

Contributors: Ari Adler, Omar Alonso, Cathy Baird, Phil Bates, Catherine Bauer, Mark Bauer, Ravinder Booreddy, Steve Cave, Steve Corbett, Claire Dessaux, Roger Ford, William Gietz, Steven Leung, Anand Manikutty, Narayan Mantravadi, Jack Melnick, Jitendra Pandey, Andy Page, Rahul Pathak, Padmini Ranganathan, Den Raphaely, Jim Rawles, David Saslav, Chitra Sharma, Keith Swartz, Priya Vennapusa, Melanie Watson, Jon Wilkinson, Vikram Yavagal, Tim Yu, Kongyi Zhou

Graphics: Valerie Moore

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent and other intellectual and industrial property laws. Reverse engineering, disassembly or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and Express, Oracle Application Server, Oracle Discoverer, Oracle Press, Oracle7, Oracle8, Oracle8i, Oracle9i, PL/SQL, Pro*C, Pro*C/C++, SQL*Net, and SQL*Plus are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

Contents

Send Us Your Comments	xxxix
Preface.....	xli
About this Guide.....	xlii
Audience	xliii
Feature Coverage and Availability	xliv
How this Manual is Organized.....	xliv
Related Documentation	l
How to Order this Manual	li
Downloading Release Notes, Installation Guides, White Papers,.....	lii
How to Access this Manual On-Line	lii
Conventions.....	lii
Documentation Accessibility	lv
What's New in Oracle XML-Enabled Technology?	lvii
Features Introduced with Oracle Application Developer's Guide - XML, Release 10g (9.0.4)	lviii
Part I Introducing Oracle XML-Enabled Technology	
1 Oracle XML-Enabled Technology	
What is XML?.....	1-2
What are Oracle XML-Enabled Technologies?.....	1-2
Oracle XML Components.....	1-2
Storing and Retrieving XML Data from Oracle	1-6

XML Support in the Database	1-7
XML and URI Data Types	1-7
Extensibility and XML	1-9
Oracle Text Searching.....	1-9
Oracle-Based XML Applications	1-10
When to Use Oracle XML Components: How They Work Together.....	1-10
Oracle XML-Enabled Technology Components and Features	1-11
Indexing and Searching XML Documents with Oracle Text (<i>interMedia</i> Text)	1-11
Messaging Hubs and Middle Tier Components	1-12
Back-End to Database to Front-End Integration Issues	1-13
Oracle XDKs Provide the Two Most Common APIs: DOM and SAX	1-13
Writing Custom XML Applications	1-14
The Oracle Suite of Integrated Tools and Components	1-14
Oracle JDeveloper and Oracle Business Components for Java (BC4J).....	1-14
Oracle Internet File System	1-15
Oracle Portal	1-15
Oracle Exchange.....	1-16
XML Gateway.....	1-16
Metadata API.....	1-16
Other XML Initiatives	1-17
Oracle XML Samples and Demos	1-17
What Is Needed to Run Oracle XML Components	1-18
Requirements for XDK.....	1-18
Which XML Components are Included with Oracle Database and Oracle Application Server? 1-18	
XML Technical Support	1-19

2 Modeling and Design Issues for Oracle XML Applications

XML Data can be Stored as Generated XML or Composed XML	2-2
Generated XML	2-2
Composed (Authored/Native) XML	2-3
Storing Composed XML Data in CLOBs or BFILEs.....	2-3
Oracle Text Indexing Enables Fine Grain Searching of Element Content	2-4
Advantages of Using Composed (Authored) XML Storage.....	2-4
Disadvantages of Using Composed XML Storage.....	2-4

Using a Hybrid XML Storage Approach for Better Mapping Granularity.....	2-5
A Hybrid Approach Allows for User-Defined Storage Granularity.....	2-5
Hybrid Storage Advantages.....	2-6
Transforming Generated XML	2-7
Combining XML Documents and Data Using Views	2-7
Indexing and Querying Transformations	2-7
Indexing Approaches.....	2-8
XML Schemas and Mapping of Documents	2-8
XMLSchema Example 1: Defining a Simple Data Type.....	2-9
XMLSchema Example 2: Map Generated XML Documents to Underlying Schema.....	2-9
General XML: Design Issues for Data Exchange Applications	2-11
Generating a Web Form from XML Data Stored in the Database.....	2-11
Sending XML Data from a Web Form to the Database.....	2-11
Sending XML Documents Applications-to-Application	2-12
Loading XML into a Database.....	2-13
Using SQL*Loader.....	2-13
Loading XML Documents Into LOBs With SQL*Loader.....	2-14
Applications that Use Oracle XML -EnabledTechnology	2-17
Content and Document Management with XML-Enabled Technology.....	2-17
Customizing Presentation of Data	2-17
Scenario 1. Content and Document Management: Publishing Composite Documents Using XML-Enabled OracleTechnology.....	2-19
Scenario 2. Content and Document Management: Delivering Personalized Information Using Oracle XML Technology	2-21
Scenario 3. Content Management: Using Oracle XML Technology to Customize Data Driven Applications.....	2-24
Business-to-Business and Business-to-Consumer Messaging.....	2-25
Scenario 4. B2B Messaging: Online Multivendor Shopping Cart Design Using XML	2-25
Scenario 5. B2B Messaging: Using Oracle XML Components and Advanced Queueing for an Online Inventory Application.....	2-27
Scenario 6. B2B Messaging: Using Oracle XML-Enabled Technology and AQ for Multi-Application Integration	2-30

3 Oracle XML Developer Kits (XDKs) and Components: Overview and General FAQs

Oracle XML Components: Overview.....	3-2
--------------------------------------	-----

Development Tools and Other XML-Enabled Features	3-3
XDK for Java	3-4
XDK for Java Beans.....	3-5
XDK for C.....	3-5
XDK for C++	3-5
XDK for PL/SQL.....	3-5
XML Parsers	3-6
XSL Transformation (XSLT) Processor.....	3-7
XML Class Generator	3-8
XML Transviewer Java Beans	3-9
Oracle XSQL Page Processor and Servlet.....	3-10
Servlet Engines that Support XSQL Servlet.....	3-11
JavaServer Pages Platforms that Support XSQL Servlet.....	3-11
Oracle XML SQL Utility (XSU)	3-14
Generating XML from Query Results.....	3-15
XML Document Structure: Columns Are Mapped to Elements.....	3-15
Oracle Text.....	3-16
Oracle XML Components: Generating XML Documents	3-17
Using Oracle XML Components to Generate XML Documents: Java	3-17
Using Oracle XML Components to Generate XML Documents: C	3-20
Using Oracle XML Components to Generate XML Documents: C++	3-22
Using Oracle XML Components to Generate XML Documents: PL/SQL	3-24
Frequently Asked Questions (FAQs): Oracle XML-Enabled Technology.....	3-26
General XDK Questions	3-26
What XML Components Do I Need to Install?.....	3-26
Building an XML Application: What Software Is Needed?.....	3-27
DTD to Database Schema	3-27
Schema Map to XML	3-28
Are There XDK Utilities That Translate From Other Formats to XML?.....	3-28
Can Oracle Generate a Database Schema From a Rational Rose Generated XML File? ..	3-29
Does Oracle Offer Any Tools to Create and Edit XML Documents?	3-29
How Can I Format XML Documents as PDF?.....	3-29
How Do I Load a Large XML Document Into the Database?	3-30
Portability and XML Support in Older Oracle Releases.....	3-32
Can I Use Parsers from Different Vendors?.....	3-32

Is There XML Support in Oracle 8.0.x?.....	3-33
Oracle 7.3.4: Data Transfers to Other Vendors Using XML	3-33
If I Use Versions Prior to Oracle8i Can I use Oracle XML Tools?	3-33
Browsers that Support XML	3-34
Which Browsers Support XML?	3-34
Standards	3-34
Are there Advantages of XML Over EDI?	3-34
What B2B Standards and Development Tools Does Oracle Support?.....	3-35
What is Oracle Corporation’s Direction Regarding XML?.....	3-36
Are There Standard DTDs that We Can Use for Orders, Shipments, and So On?	3-37
XML, CLOBs, and BLOBs	3-37
Is There Support for XML Messages in BLOBs?	3-37
Maximum FileSizes	3-37
What is the Maximum XML File Size When Stored in CLOBs?	3-37
XML File Size Limitations	3-37
Maximum Size for an XML Document.....	3-38
Inserting XML Data Into Tables	3-38
What Do I Need to Insert Data Into Tables Using XML?	3-38
XML in the Database: Performance	3-39
Where Can I Find Information about the Performance of XML and Oracle?.....	3-39
How Can I Speed Up the Record Retrieval in XML Documents?.....	3-39
Using XML With Different Languages	3-39
Further References	3-40
Other XML Frequently Asked Questions	3-40
Recommended XML and XSL Books.....	3-40

4 Using XSL and XSLT

Introducing XSL	4-2
The W3C XSL Specification.....	4-2
Namespaces in XML	4-3
XSL Stylesheet Architecture.....	4-3
XSL Transformation (XSLT)	4-4
XSLT 1.1 Specification.....	4-4
XML Path Language (Xpath)	4-5
CSS Versus XSL	4-5

XSL References	4-6
Frequently Asked Questions: XSL and XSLT	4-6
How Do I Write an IF Statement in XSL That Tests for Values Within Tags?	4-6
In an XSL Document, How Can We Select Specific Attributes?	4-7
When Converting XML to HTML, Why Do I get "Unexpected EOF"?	4-7
Whitespace: Why are my Resulting Values Multiplied by 2?	4-8
How Can I Specify a NULL Indicator in XSL?	4-10
How Can Transfer Tag Names in XSLT?	4-10
How Do I Convert A String to a Nodeset in XSL?	4-13
In XSL, How Can I Correctly Convert an XML Document Tag to a Link in HTML?	4-16
Am I Using the Correct XSL Headers for my WML Transformation?	4-17
In an XSL Transformation, How Do I Ensure that the DTD File Can be Located?	4-18
In XSL, How to Prevent the Namespace Definition from Being Repeated	4-19
How Do I Pass a Parameter from a Java Program to an XSL Stylesheet?	4-20
How Can I Resolve the Error XSL-1009 Attribute 'XSL Version' Not Found in HTML?	4-21
What XPath Expression Will Retrieve Only Terminal Child Elements?	4-22
Child Attributes are Not Returned After Applying XSL Stylesheet	4-23

Part II Storing and Retrieving XML From the Database

5 Database Support for XML

What are the Oracle Native XML Database Features?	5-2
XMLType Datatype	5-3
How to use XMLType	5-4
Guidelines for using XMLType Columns	5-6
Benefits of XMLType	5-7
When to use XMLType	5-8
XMLType Storage in the Database	5-8
Specifying Storage Characteristics on XMLType Columns.....	5-10
Specifying Constraints on XMLType Columns.....	5-12
XMLType Functions	5-13
Manipulating XML Data in XMLType Columns	5-16
Inserting XML Data into XMLType Columns	5-16
Updating XML Data in XMLType Columns	5-18
Deleting XML Data.....	5-19

Using XMLType Inside Triggers.....	5-19
Selecting and Querying XML Data	5-20
Selecting XML data.....	5-20
Querying XML data	5-21
Querying XMLType Data using Text Operators	5-29
Indexing XMLType columns	5-30
Java Access to XMLType (oracle.xdb.XMLType)	5-32
Installing and using oracle.xdb.XMLType class	5-40
Native XML Generation	5-41
DBMS_XMLGEN	5-41
SYS_XMLGEN	5-62
XMLGenFormatType Object.....	5-63
SYS_XMLAGG	5-71
Other Aggregation Methods.....	5-76
TABLE Functions	5-76
Using Table Functions with XML	5-77
Table Functions Example 1: Exploding the PO to Store in a Relational Table	5-77
Frequently Asked Questions (FAQs): XMLType	5-80

6 Database Uri-references

Uri-reference (Uri-ref) Concepts	6-2
What is a Uri-ref?.....	6-2
Advantages of Using DBUri-ref	6-3
New Datatypes Store Uri-references	6-3
Benefits of Using UriTypes	6-4
DBUri-refs, Intra-Databases References	6-4
Formulating the DBUri	6-5
The DB-Uri Specification	6-7
DBUri Syntax Guidelines	6-8
Some Common DBUri-ref Scenarios	6-9
How DBUri's Differ from Object References.....	6-12
DBUri-ref Applies to a Database and Session	6-12
Where Can DBUri-ref be Used?.....	6-12
Using Uri-ref Types (URITypes)	6-14
Storing Pointers to Documents with UriType.....	6-14

URIType Examples	6-15
Using HttpUriType and DBUriType	6-16
DBUriType Examples.....	6-16
UriFactory Package	6-17
UriFactory Example: Registering the ecom Protocol.....	6-18
Why Use Different Uri-refs?	6-19
SYS_DBURIGEN() SQL Function	6-20
SYS_DBURIGEN Example 1: Inserting Database References	6-22
SYS_DBURIGEN Example 2: Returning Partial Results	6-22
SYS_DBURIGEN Example 3: RETURNING Uri-refs	6-24
Accessing DBUri-refs From Your Browser Using Servlets	6-25
oracle.xml.dburi.OraDbUriServlet() Servlet Mechanism.....	6-25
OraDBUriServlet Security	6-26
Installing OraDBUri Servlet	6-27
DBUri Servlet Example 1: First Create a DBUriServer Web Service [tkxmsrv.ssh]	6-28
DBUri Servlet Example 2: Creating DBUridomain — Publishing OraDbUriServlet.....	6-29
DBUri Servlet Example 3: Publishing OraDbUriServlet Under SYS [tkxmsysd.ssh]	6-30
DBUri Servlet Example 4: Publishing OraDbUriServlet Under ADAMS	6-31
DBUri Servlet Example 5: Publishing OraDbUriServlet Under SCOTT [tkxmsctd.ssh]..	6-32
DBUri Servlet Example 6: Creating and Mapping dburirealm — OraDbUriServlet.....	6-33
DBUri Servlet Example 7: Publishing OraDbUriServlet Under the ADAMS Schema	6-34
DBUri Servlet Example 8: Publishing OraDbUriServlet Under the ADAMS Schema	6-35
Configuring the UriFactory Package to Handle DBUri-refs	6-36

7 XML SQL Utility (XSU)

What is XML SQL Utility (XSU)?	7-2
XSU Features	7-3
XSU Oracle Features.....	7-3
XSU Dependencies and Installation	7-4
Dependencies	7-4
Installation	7-4
XML SQL Utility and the Bigger Picture	7-5
XML SQL Utility in the Database	7-5
XML SQL Utility in the Middle Tier	7-6
XML SQL Utility in a Web Server	7-7

XML SQL Utility In The Client Tier	7-8
SQL-to-XML and XML-to-SQL Mapping Primer	7-8
Default SQL-to-XML Mapping	7-8
Customizing the Generated XML: Mapping SQL to XML	7-12
Default XML-to-SQL Mapping	7-13
How XML SQL Utility Works	7-14
Selecting with XSU	7-14
Inserting with XSU	7-14
Updating with XSU	7-15
Deleting with XSU	7-16
Using the XSU Command Line Front End,OracleXML	7-17
Generating XML Using the XSU Command Line	7-17
XSU's OracleXML getXML Options	7-18
Inserting XML Using XSU's Command Line (putXML).....	7-19
XSU OracleXML putXML Options.....	7-20
XSU Java API	7-20
Generating XML with XSU's OracleXMLQuery	7-21
Generating XML From SQL Queries Using XSU	7-21
XSU Generating XML Example 1: Generating a String From Table emp (Java)	7-22
XSU Generating XML Example 2: Generating DOM From emp table (Java)	7-25
Paginating Results: skipRows and maxRows	7-27
Keeping the Object Open For the Duration of the User's Session.....	7-27
When the Number of Rows or Columns in a Row Are Too Large	7-27
keepObjectOpen Function	7-28
XSU Generating XML Example 3. Paginating Results: Generating an XML Page (Java)	7-28
Generating XML from ResultSet Objects	7-30
XSU Generating XML Example 4: Generating XML from JDBC ResultSets (Java).....	7-30
XSU Generating XML Example 5: Generating XML from Procedure Return Values	7-32
Raising No Rows Exception	7-33
XSU Generating XML Example 6: No Rows Exception (Java)	7-34
Storing XML Back in the Database Using XSU OracleXMLSave	7-35
Insert Processing Using XSU (Java API)	7-36
XSU Inserting XML Example 7: Inserting XML Values into All Columns (Java)	7-36
XSU Inserting XML Example 8: Inserting XML Values into Only Certain Columns	7-37
Update Processing Using XSU (Java API)	7-38

XSU Updating XML Example 9: Updating a Table Using the keyColumns (Java)	7-39
XSU Updating XML Example 10: Updating a Specified List of Columns (Java)	7-40
Delete Processing Using XSU (Java API)	7-41
XSU Deleting XML Example 11: Deleting Operations Per ROW (Java)	7-41
XSU Deleting XML Example 12: Deleting Specified Key Values (Java)	7-42
XSU PL/SQL API	7-43
Generating XML with DBMS_XMLQuery()	7-43
XSU Generating XML Example 13: Generating XML From Simple Queries (PL/SQL) ..	7-43
XSU Generating XML Example 13a: Printing CLOB to Output Buffer	7-44
XSU Generating XML Example 14: Changing ROW and ROWSET Tags (PL/SQL)	7-44
XSU Generating XML Example 15: Using setMaxRows() and setSkipRows()	7-45
Setting Stylesheets in XSU (PL/SQL)	7-46
Binding Values in XSU (PL/SQL)	7-47
XSU Generating XML Example 15a: Binding Values to the SQL Statement.....	7-47
Storing XML in the Database Using DBMS_XMLSave	7-48
Insert Processing Using XSU (PL/SQL API)	7-49
XSU Inserting XML Example 16: Inserting Values into All Columns (PL/SQL)	7-49
XSU Inserting XML Example 17: Inserting Values into Certain Columns (PL/SQL)	7-50
Update Processing Using XSU (PL/SQL API)	7-51
XSU Updating XML Example 18: Updating an XML Document Using keyColumns	7-52
XSU Updating XML Example 19: Specifying a List of Columns to Update (PL/SQL)	7-52
Delete Processing Using XSU (PL/SQL API)	7-53
XSU Deleting XML Example 20: Deleting Operations per ROW (PL/SQL)	7-53
XSU Example 21: Deleting by Specifying the Key Values (PL/SQL)	7-54
XSU Deleting XML Example 22: ReUsing the Context Handle (PL/SQL)	7-55
Advanced XSU Usage Techniques	7-56
XSU Exception Handling in Java	7-56
XSU Exception Handling in PL/SQL	7-58
Frequently Asked Questions (FAQs): XML SQL Utility (XSU)	7-58
What Schema Structure Should I Use With XSU to Store XML?	7-58
Can XSU Store XML Data Across Tables?	7-60
Can I Use XML SQL Utility to Load XML Stored in Attributes?	7-61
Is XML SQL Utility Case Sensitive? Can I Use ignoreCase?	7-61
Will XSU Generate Database Schema from a DTD?	7-61
Can You Provide a Thin Driver Connect String Example for XSU?	7-62

Does XML SQL Utility Commit After INSERT, DELETE, UPDATE?	7-62
Can You Explain How to Map Table Columns to XML Attributes Using XSU?	7-63
How Can I Use XMLGEN.insertXML with LOBs?	7-64

8 Searching XML Data with Oracle Text

Introducing Oracle Text	8-3
Assumptions Made in this Chapter's Examples	8-4
Oracle Text Users and Roles	8-5
Querying with the CONTAINS Operator	8-6
Using a Simple SELECT Statement	8-6
Using the Score Operator with a Label to Obtain the Relevance	8-7
Using the WITHIN Operator to Narrow Query Down to Document Sections	8-7
Using INPATH or HASPATH Operators for <i>Query Searching</i>	8-10
Using Oracle Text to Search XML Documents	8-17
Step 1. Create a Section Preference	8-18
Step 2. Create an Index Using the Section Preference Created in Step 1	8-21
Oracle Text Example 1: Creating an Index Using XML_SECTION_GROUP	8-21
Oracle Text Example 2: Creating an Index Using AUTO_SECTION_GROUP	8-23
Oracle Text Example 3: Creating an Index Using PATH_SECTION_GROUP	8-23
Building XML Query Applications with Oracle Text	8-23
Querying XML Documents	8-24
Distinguishing Tags Across DocTypes	8-24
Specifying Doctype Limiters to Distinguish Between Tags	8-24
Doctype-Limited and Unlimited Tags in a Section Group	8-25
Querying Within Attribute Sections	8-25
XML_SECTION_GROUP Attribute Sections	8-26
Constraints for Querying Attribute Sections	8-28
Procedure for Building a Query Application with Oracle Text	8-29
Using Table CTX_OBJECTS and CTX_OBJECT_ATTRIBUTES View	8-30
Step 1. Create a Preference	8-31
Step 2. Set the Preference's Attributes	8-31
2.1 Using CTX_DDL.add_zone_section	8-31
2.2 Using CTX_DDL.Add_Attr_Section	8-32
2.3 Using CTX_DDL.add_field_section	8-33
2.5 Using CtX_DDL.Add_Stop_Section	8-35

Step 3. Create Your Query Syntax	8-35
Oracle Text Example 4: Querying a... Document.....	8-36
Oracle Text Example 5: Creating an Index and Performing a Text Query.....	8-38
Creating Sections in XML Documents that are Document Type Sensitive	8-39
Repeated Sections	8-39
Overlapping Sections	8-40
Nested Sections	8-40
Presenting the Results of Your Query	8-41
Case Study: Searching an Online FAQ List Using Oracle Text	8-41
1 Create and Populate Your FAQ Table. Create an Auto Section Group and Text Index	8-44
2 Compile showxml.psp	8-46
3 Compile faqsearch.psp.....	8-47
Frequently Asked Questions (FAQs): Oracle Text	8-50
Searching Attribute Values	8-50
Can I Build Indexes on Attribute Values?.....	8-50
General Oracle Text Questions	8-51
Can XML Documents Be Queried Like Table Data?	8-51
Can we Search Based on Structural Conditions?	8-52
How Can I Searching XML Documents and Return a Zone?.....	8-52
Loading XML Documents into the Database and Searching with Oracle Text.....	8-53
How Do I Search XML using the WITHIN Operator?	8-53
Oracle Text (intermedia Text) and XML.....	8-54
Oracle Text (intermedia Text) and XML: Add_field_section	8-54
Can I Do Range Searching with Oracle Text?.....	8-55
Can Oracle Text Do Section Extraction?.....	8-55
Can I Create a Text Index on Three Columns?.....	8-55
How Fast is Oracle at Indexing Text and Can I Just Enable Boolean Searches?.....	8-56
How Can We Index XML Documents in Different Languages?.....	8-57
Searching XML Documents in CLOBs	8-58
How Do I Search CLOBs Using Oracle Text?.....	8-58
How Can I Search Different XML Documents Stored in CLOBs With Different DTDs?.	8-58
Storing an XML Document in CLOB: Using Oracle Text (intermedia Text).....	8-60
Can We Only Insert Structured When The Table is Created?.....	8-61

Part III Data Exchange Using XML

9 Exchanging XML Data Using Oracle AQ

What is AQ?	9-2
How do AQ and XML Complement Each Other?	9-2
Internet-Data-Access-Presentation (IDAP)	9-6
XML and the IDAP Interface	9-6
IDAP Architecture	9-6
IDAP Method Invocation	9-8
IDAP Message Structure.....	9-9
IDAP Method Invocation Body: “IDAP Payload”	9-10
IDAP Message Body is an AQ XML Document	9-11
IDAP Client Requests for Enqueue	9-11
Message Payloads.....	9-13
IDAP Enqueue Request Example 1 — ADT Message to a Single-Consumer Queue	9-17
IDAP Enqueue Request Example 2 — Message to a Multiconsumer Queue	9-19
IDAP Enqueue Request Example 3 — Sending a Message to a JMS Queue.....	9-20
IDAP Enqueue Request Example 4 — Sending/Publishing and Committing	9-21
IDAP Client Requests for Dequeue	9-22
IDAP Dequeue Request Example 1— Messages from a Single-Consumer Queue	9-24
IDAP Dequeue Request Example 2 — Messages that Satisfy a Specific Condition	9-25
IDAP Dequeue Request Example 3 — Receiving Messages and Committing.....	9-25
IDAP Dequeue Request Example 4 — Browsing Messages.....	9-26
IDAP Client Requests for Registration	9-26
IDAP Register Request Example 1— Registering for Notification at an Email Address.	9-27
Commit Request	9-27
Rollback Request.....	9-28
IDAP Server Response to Enqueue	9-28
IDAP Server Request Example 1 — Enqueuing to a Single-Consumer Queue.....	9-29
IDAP Server Request Example 2— Enqueuing to a Multiconsumer Queue	9-29
Server Response to a Dequeue Request	9-30
IDAP Server Dequeue Response Example 1 — Messages from an ADT Queue	9-30
Server Response to a Register Request	9-31
Commit Response.....	9-31
Rollback Response	9-32
Notification	9-32
IDAP and AQ XML Schemas	9-33

AQXMLServlet	9-33
Accessing AQXMLServlet with HTTP	9-33
XMLType Queues	9-37
Storing and Querying XML Documents with Advanced Queueing (AQ).....	9-37
Structuring and Managing Message Payloads with Object Types.....	9-37
Creating Message Payloads Queues Containing XMLType Attributes	9-37
XMLType Queues Example 1: Creating XMLType Queue Tables for a Queue Object....	9-38
AQ XML Message Format Transformation	9-38
AQ Message Transformation Example 1: Creating a PL/SQL Function.....	9-39
Frequently Asked Questions (FAQs): XML and Advanced Queueing	9-41
Can we Store AQ XML Messages with Many PDFs as One Record?	9-41
Can We Add New Recipients After Messages are Enqueued?.....	9-42
How Does Oracle Enqueue and Dequeue and Process XML Messages?	9-43
How Can We Parse Messages with XML Content From AQ Queues?.....	9-44
Can we Prevent the Listener From Stopping Until the XML Document is Processed? ...	9-45

Part IV Tools and Frameworks for Building Oracle-Based XML Applications

10 XSQL Pages Publishing Framework

XSQL Pages Publishing Framework Overview	10-2
What Can I Do with Oracle XSQL Pages?	10-3
Where Can I Obtain Oracle XSQL Pages?	10-5
What's Needed to Run XSQL Pages?.....	10-5
Overview of Basic XSQL Pages Features	10-6
Producing XML Datagrams from SQL Queries	10-7
Transforming XML Datagrams into an Alternative XML Format	10-9
Transforming XML Datagrams into HTML for Display	10-13
Setting Up and Using XSQL Pages in Your Environment	10-15
Using XSQL Pages With Oracle JDeveloper	10-15
Setting the CLASSPATH Correctly in Your Production Environment	10-16
Setting Up the Connection Definitions.....	10-17
Using the XSQL Command Line Utility.....	10-18
Overview of All XSQL Pages Capabilities	10-19
Using All of the Core Built-in Actions.....	10-19
Aggregating Information Using <xsql:include-xsql>	10-36

Handling Posted Information.....	10-38
Using Custom XSQL Action Handlers.....	10-44
Description of XSQL Servlet Examples.....	10-46
Setting Up the Demo Data.....	10-48
Advanced XSQL Pages Topics.....	10-49
Understanding Client Stylesheet-Override Options.....	10-49
Controlling How Stylesheets are Processed.....	10-50
Using XSQLConfig.xml to Tune Your Environment.....	10-54
Using the FOP Serializer to Produce PDF Output.....	10-59
Using XSQL Page Processor Programmatically.....	10-61
Writing Custom XSQL Action Handlers.....	10-63
Writing Custom XSQL Serializers.....	10-68
Writing Custom XSQL Connection Managers.....	10-71
Formatting XSQL Action Handler Errors.....	10-72
XSQL Servlet Limitations.....	10-73
HTTP Parameters with Multibyte Names.....	10-73
CURSOR() Function in SQL Statements.....	10-73
Frequently Asked Questions (FAQs) - XSQL Servlet.....	10-74
Specifying a DTD While Transforming XSQL Output to a WML Document.....	10-74
XSQL Servlet Conditional Statements.....	10-74
Using Value Retrieved in One Query in Another Query's Where Clause.....	10-75
Working with Non-Oracle Databases.....	10-75
XSQL Servlet: Access to JServ Process.....	10-76
XSQL on Oracle8i Lite.....	10-76
Handling Multi-Valued HTML Form Parameters.....	10-77
XSQL Servlet and Oracle 7.3.....	10-79
Out Variable Not Supported in <xsql:dml>.....	10-79
Unable to Connect Errors.....	10-81
Using Other File Extensions Besides *.xsql.....	10-81
Avoiding Errors for Queries Containing XML Reserved Characters.....	10-82

11 Using JDeveloper to Build Oracle XML Applications

Introducing JDeveloper9i.....	11-2
Business Components for Java (BC4J).....	11-3
Oracle JDeveloper XML Strategy.....	11-4

Further Information About JDeveloper.....	11-5
What's Needed to Run JDeveloper9i.....	11-5
Accessing JDeveloper9i.....	11-6
XML in Business Components for Java (BC4J)	11-6
Building XSQL Clients with Business Components for Java (BC4J)	11-8
Object Gallery.....	11-8
XSQL Element Wizard	11-10
Page Selector Wizard.....	11-11
XML Features in JDeveloper9i	11-11
Oracle XDK and Transviewer Beans Integration	11-11
Oracle XML Parser for Java	11-12
Oracle XSQL Servlet	11-12
XML Data Generator Web Bean	11-14
Mobile Application Development with Portal-To-Go and JDeveloper	11-14
Building XML Applications with JDeveloper.....	11-14
JDeveloper XML Example 1: BC4J Metadata.....	11-14
Procedure for Building Applications in JDeveloper9i.....	11-15
Using JDeveloper's XML Data Generator Web Bean.....	11-15
Using XSQL Servlet from JDeveloper	11-17
JDeveloper XSQL Example 2: Employee Data from Table emp: emp.xsql	11-18
JDeveloper XSQL Example 3: Employee Data with Stylesheet Added	11-19
Creating a Mobile Application in JDeveloper	11-20
1 Create the BC4J Application.....	11-21
2 Create JSP Pages Based on a BC4J Application	11-22
3 Create XSLT Stylesheets According to the Devices Needed to Read The Data	11-23
Frequently Asked Questions (FAQs): Using JDeveloper to Build XML Applications	11-26
Constructing an XML Document in JSP	11-26
Using XMLData From BC4J	11-27
Running XML Parser for Java in JDeveloper 3.0.....	11-28
Moving Complex XML Documents to a Database	11-31

12 Building BC4J and XML Applications

Introducing Business Components for Java (BC4J)	12-2
BC4J Features	12-2
BC4J Advantages	12-3

Building BC4J XML Applications in JDeveloper	12-4
Building XSQL Clients with BC4J	12-5
Ease of Code Generation and Management when Building XML and Java Applications	12-6

13 Using Metadata API

Introduction to Metadata API	13-2
Previous Methods Used to Extract Metadata	13-2
Metadata API Components	13-2
Metadata API Features	13-3
Internet Computing	13-4
What is DBMS_METADATA?	13-4
DBMS_METADATA and Security	13-5
DBMS_METADATA Programmatic Interface	13-6
DBMS_METADATA.FETCH_XML	13-9
DBMS_METADATA.FETCH_DDL()	13-10
Performance Tips	13-13
DBMS_METADATA Browsing Interface	13-13
Metadata API Example: Retrieving DDL for Tables	13-15
mddemo.sql	13-15
PAYROLL_DEMO Output	13-21

14 OracleAS Reports Services and XML

Introducing OracleAS Reports Services and XML	14-2
B2B Data Exchange: Why Use XML in Reports?	14-2
What's Needed to Run OracleAS Reports Services	14-4
Creating XML Output "On the Fly" Using OracleAS Reports Services	14-4
XML as a Data InterChange Format	14-4
Formatting XML Output Using XSL Stylesheets	14-5
Customizing Report Definitions at Runtime	14-6
Applying an XML Customization	14-6
Customizing Reports at Runtime with XML	14-8
Customizing Reports with XML, Example 1: Modifying F_EMPNO and Setting Color.	14-9
Customizing Reports with XML, Example 2: Changing Text Color of F_EMPNO	14-9
Customizing Reports with XML, Example 3: Modifying Boilerplate Text Objects	14-10

Customizing Reports with XML, Example 4: Replacing a SELECT * Query	14-10
Customizing Reports with XML, Example 5: Adding a Trigger to Field S_SAL	14-11
Performing Batch Report Modifications by Applying XML Report Definitions	14-12
Creating Mutated RDFs Out of One Master	14-13
Creating Multi-Version Reports Out of a Single RDF	14-13
Customizing Reports with XML Example 6: Creating Different Language Versions....	14-13
Creating Report Definitions in XML.....	14-15
Customizing Reports with XML, Example 7: Report from XML Definitions Only	14-15
Running XML Report Definitions	14-16
Running an XML Report Definition by Itself	14-16
XML Used in JSP for Storing Report Definitions	14-17
Using XML as a Datasource	14-17
Pluggable Data Source, XML-PDS	14-17
Using XML for OracleAS Reports Services Configuration Files.....	14-18
How Reports9i XML-PDS Supports XSQL Servlet	14-19
Reports Case Studies	14-20
How to Become a Supplier of Live XML Streams.....	14-20
How to Take Advantage of Supplied XML-Data.....	14-21
Frequently Asked Questions: Reports and XML.....	14-24
Can We Output XML From Our Year End Reports Through a Database Interface?.....	14-24
Changing the Report Template.....	14-25
REP-6106:Error in the XML report definition at line 1	14-26

15 Using the PDK for Visualizing XML Data in Oracle Portal

Introducing Oracle Portal.....	15-2
What are Portlets?.....	15-2
Common Portlet Applications	15-3
Oracle Portal Development Kit (PDK)	15-3
PDK Integration Services (PDKIS)	15-4
PDK URL Services	15-4
What's Needed to Run URL Services	15-4
PDK URL Services Overview	15-4
Creating a URL Portlet.....	15-5
Web Provider.....	15-5
URL Services Architecture	15-5

URL Services Interface	15-6
URL Services Runtime	15-6
Provider.xml	15-7
Using provider.xml	15-8
Configuring provider.xml	15-9
Provider Tag	15-9
Portlet Tag	15-9
Integrating Technologies into OracleAS Portal	15-14

16 How Oracle Exchange Uses XML

Oracle Exchange and XML	16-2
Stored Transactions	16-2
Pass Through Transactions	16-3
XML Delivery Formats	16-4
E-Business Solution Architecture	16-4
ATP (Availability to Promise) for Oracle Exchange	16-5
The webMethods Services	16-5
Exchange - Supplier XML	16-5
XML Messaging Services	16-9
XML Message Designer and Runtime Execution Engine	16-9
Generating XML that Conforms to New Schema	16-9

17 Introducing Oracle XML Gateway

What is XML Gateway?	17-2
Oracle XML Gateway Services	17-2
Oracle XML Gateway Architecture	17-3
XML Gateway Services - Message Designer	17-4
XML Gateway Services - Message Set Up	17-7
XML Gateway Services - Execution Engine	17-9
A Word About XML Standards	17-10

Part V OracleAS Dynamic Services (DS) and Oracle Syndication Server (OSS)

18 Using OracleAS Dynamic Services and XML

Introducing OracleAS Dynamic Services	18-2
How Dynamic Services (DS) Helps Developers	18-3
For Further Information.....	18-4
What is Needed to Run OracleAS Dynamic Services?	18-4
Dynamic Services (DS) Architecture Overview	18-4
Dynamic Services (DS) Implementation Overview	18-6
Dynamic Services Java Deployment	18-7
Dynamic Services PL/SQL Deployment	18-8
Dynamic Services Java HTTP/Java Messaging Services (JMS) Deployment	18-9
Multiple Channel Capabilities of DS	18-11
Dynamic Services Features	18-12
Service Management and Administration	18-12
Service Discovery.....	18-13
Service Execution	18-13
Dynamic Services Integrates with Other Oracle Products	18-16
How Service Consumers Use Dynamic Services	18-17
Developing Services For Dynamic Services	18-17
Oracle Syndication Server (OSS)	18-18
Dynamic Services Consumer Application: Stock Portfolio Example	18-18
Compiling SampleStock.java	18-19
Dynamic Services Example 1: SampleStock (Java)	18-21
Frequently Asked Questions (FAQs): Dynamic Services	18-27
How to Set Up a Language of Queuing and Sequencing Commands?.....	18-27
Other FAQs?	18-27

19 Oracle Syndication Server (OSS) and XML

Introducing Oracle Syndication Services (OSS)	19-2
OSS Features: e-Business Content Aggregation, Exchange, and Syndication	19-2
Content Syndication	19-3
Information and Content Exchange (ICE) Protocol	19-4
OSS Architecture	19-5
Interacting with Content Providers	19-7
Dynamic Services Content Provider Adapter (DSCPA)	19-7
Interacting With Content Subscribers	19-7

Delivering content to subscribers.....	19-8
--	------

Part VI XDK for Java

20 Using XML Parser for Java

XML Parser for Java: Features	20-2
XSL Transformation (XSLT) Processor	20-4
Namespace Support	20-5
Oracle XML Parsers Support Four Validation Modes	20-5
Parsers Access XML Document's Content and Structure	20-6
DOM and SAX APIs	20-8
DOM: Tree-Based API.....	20-8
SAX: Event -Based API	20-8
Guidelines for Using DOM and SAX APIs.....	20-9
XML Parser and Data Compression	20-10
XML Serialization/Compression	20-10
Upgrading XDK for Java	20-11
Upgrading XDK for Java from a Previous Release to Oracle.....	20-11
Downgrading to Oracle Release 8.1	20-12
Running the XML Parser for Java Samples	20-12
XML Parser for Java - XML Sample 1: class.xml.....	20-13
XML Parser for Java - XML Example 2: Using DTD employee — employee.xml	20-14
XML Parser for Java - XML Example 3: Using DTD family.dtd — family.xml.....	20-14
XML Parser for Java — XSL Example 1: XSL (iden.xml)	20-15
XML Parser for Java - DTD Example 1: (NSExample)	20-15
Using XML Parser for Java: DOMParser() Class	20-16
XML Parser for Java Example 1: Using the Parser and DOM API (DomSample.java) ..	20-18
Comments on DOMParser() Example 1	20-22
Using XML Parser for Java: DOMNamespace() Class	20-23
XML Parser for Java Example 2: Parsing a URL — DOMNamespace.java	20-23
Using XML Parser for Java: SAXParser() Class	20-27
XML Parser for Java Example 3: Using the Parser and SAX API (SAXSample.java)	20-29
Using XML Parser for Java: XSLT Processor	20-33
XML Parser for Java Example 4: (XSLSample.java).....	20-35
XML Parser for Java Example 5: Using the DOM API and XSLT Processor	20-38

Comments on XSLT Example 5	20-40
Using XML Parser for Java: SAXNamespace() Class	20-41
XML Parser for Java Example 6: (SAXNamespace.java)	20-41
XML Parser for Java: Command Line Interface	20-45
oraxml - Oracle XML parser.....	20-45
oraxsl - Oracle XSL processor	20-46
XML Extension Functions for XSLT Processing	20-47
XSLT Processor Extension Functions: Introduction	20-47
Static Versus Non-static Methods	20-48
Constructor Extension Function.....	20-48
Return Value Extension Function.....	20-49
Datatypes Extension Function	20-50
ora XSLT Built In Extensions: ora:node-set and ora:output	20-50
Frequently Asked Questions (FAQs): XML Parser for Java	20-55
DTDs	20-55
Checking DTD Syntax: Suggestions for Editors.....	20-55
DTD File in DOCTYPE Must be Relative to XML Document Location.....	20-57
Validating an XML File Using External DTD	20-57
DTD Caching.....	20-57
Recognizing External DTDs	20-58
Loading external DTD's from a jar File	20-59
Can I Check the Correctness of an XML Document Using their DTD?.....	20-59
Parsing a DTD Object Separately from XML Document	20-60
Case-Sensitivity in Parser Validation against DTD?	20-60
Extracting Embedded XML From a CDATA Section.....	20-61
Why Am I Getting an Error When I Call DOMParser.parseDTD()?.....	20-62
What Is Standard Extension for External Entities References in an XML Document?...	20-64
DOM and SAX APIs	20-65
Using the DOM API	20-65
How DOM Parser Works.....	20-65
Creating a Node With Value to be Set Later.....	20-65
Traversing the XML Tree.....	20-66
Extracting Elements from XML File.....	20-66
Does a DTD Validate the DOM Tree?.....	20-66
First Child Node Element Value.....	20-67

Creating DocType Node.....	20-67
XMLNode.selectNodes() Method	20-67
Using SAX API to Get the Data Value.....	20-68
SAXSample.java	20-69
Does DOMParser implement Parser interface	20-69
Creating an New Document Type Node Via DOM	20-69
Querying for First Child Node's Value of a Certain Tag.....	20-70
XML Document Generation From Data in Variables.....	20-71
Printing Data in the Element Tags: DOM API	20-71
Building XML Files from Hashtable Value Pairs.....	20-72
XML Parser for Java: wrong_document_err on Node.appendChild()	20-72
Creating Nodes: DOMException when Setting Node Value	20-74
With SAX, How Can I Force the Parser to Not Discard Whitespace?	20-74
Validation	20-75
DTD: Understanding DOCTYPE and Validating Parser.....	20-75
Can Multiple Threads Use Single XSLProcessor/Stylesheet?	20-75
Is it Safe to Use Document Clones in Multiple Threads?	20-76
Character Sets	20-76
Encoding iso-8859-1 in xmlparser	20-76
Parsing XML Stored in NCLOB With UTF-8 Encoding.....	20-77
NLS support within XML.....	20-78
UTF-16 Encoding with XML Parser for Java V2	20-79
How Can I Read in Accented Characters?.....	20-80
Adding XML Document as a Child	20-81
Adding an XMLDocument as a Child to Another Element	20-81
Adding an XML DocumentFragment as a Child to XMLDocument	20-82
Uninstalling Parsers	20-83
Removing XML Parser from the Database	20-83
XML Parser for Java: Installation	20-84
XMLPARSER Fails to Install	20-84
General XML Parser Related Questions	20-84
How the XML Parser Works.....	20-84
Converting XML Files to HTML Files	20-85
Does XML Parser Validate Against XML Schema?	20-85
Including Binary Data in an XML Document.....	20-85

What is XML Schema?	20-86
Oracle's Participation in Defining the XML/SQL Standard	20-86
XDK Version Numbers	20-86
Inserting <, >, >= and <= in XML Documents	20-87
Are Namespace and Schema Supported	20-87
Using JDK 1.1.x with XML Parser for Java v2	20-87
Sorting the Result on the Page	20-87
Is Oracle Needed to Run XML Parser for Java?	20-88
Dynamically Setting the Encoding in an XML File.....	20-88
Parsing a String	20-88
Displaying an XML Document	20-89
System.out.println() and Special Characters	20-89
Obtaining Ampersand from Character Data	20-89
How Can We Use Special Characters in the Tags?	20-90
Parsing XML from Data of Type String.....	20-91
Extracting Data from XML Document into a String.....	20-91
Disabling Output Escaping	20-92
Using the XML Parser for Java with Oracle 8.0.5.....	20-92
Delimiting Multiple XML Documents.....	20-92
XML and Entity-references: XML Parser for Java.....	20-93
Can I Break up and Store an XML Document without a DDL Insert?.....	20-93
Merging XML Documents.....	20-94
Getting the Value of a Tag.....	20-96
Granting JAVASYSPRIV to User.....	20-96
Including an External XML File in Another XML File: External Parsed Entities.....	20-97
Where Can I Download OraXSL, The Parser's Command Line Interface?.....	20-99
Will Oracle Support Hierarchical Mapping?	20-99
XSLT Processor and XSL Stylesheets	20-100
HTML Error in XSL	20-100
Is <xsl:output method="html"/> Supported?	20-101
Netscape 4.0: Preventing XSL From Outputting <meta> Tag.....	20-103
XSL Error Messages.....	20-104
Generating HTML: "<" Character.....	20-104
HTML "<" Conversion Works in oraxsl but not XSLSample.java?	20-105
XSLT Examples	20-106

XSLT Features	20-106
Using XSL To Convert XML Document To Another Form	20-107
Information on XSL?	20-108
XSLProcessor and Multiple Outputs?	20-109
What Good Books for XML/XSL Can You Recommend?	20-109
XML Developer Kits for HP/UX Platform	20-110
Compressing Large Volumes of XML Documents	20-110
How Can I Generate an XML Document Based on Two Tables?.....	20-111

21 Using XML Schema Processor for Java

Introducing XML Schema	21-2
How DTDs and XML Schema Differ.....	21-2
XML Schema Features.....	21-3
Oracle XML Schema Processor for Java Features	21-6
Supported Character Sets	21-6
What's Needed to Run XML Schema Processor for Java.....	21-7
XML Schema Processor for Java Directory Structure.....	21-8
XML Schema Processor for Java Usage	21-8
How to Run the XML Schema for Java Sample Program.....	21-9
MakeFile.....	21-10
XML Schema for Java Example 1: cat.xsd	21-11
XML Schema for Java Example 2: catalogue.xml.....	21-12
XML Schema for Java Example 3: catalogue_e.xml.....	21-12
XML Schema for Java Example 4: report.xml.....	21-13
XML Schema for Java Example 5: report.xsd	21-14
XML Schema for Java Example 6: report_e.xml.....	21-15
XML Schema for Java Example 7: XSDSample.java	21-16
XML Schema for Java Example 8: XSDSetSchema.java	21-18

22 XML Class Generator for Java

Accessing XML Class Generator for Java.....	22-2
XML Class Generator for Java: Overview.....	22-2
Oracg Command Line Utility	22-3
Class Generator for Java: XML Schema.....	22-4
Namespace Features	22-4

Using XML Class Generator for Java with XML Schema	22-5
Generating Top Level Element Classes	22-6
Generating Top Level ComplexType Element Classes	22-7
Generating SimpleType Element Classes.....	22-7
Using XML Class Generator for Java with DTDs.....	22-8
Examples Using XML Java Class Generator with DTDs and XML Schema	22-10
Running XML Class Generator for Java — DTD Examples	22-10
Running XML Class Generator for Java — XML Schema Examples	22-11
XML Class Generator for Java, DTD Example 1a: Application — SampleMain.java.....	22-12
XML Class Generator for Java, DTD Example 1b: DTD Input — widl.dtd	22-14
XML Class Generator for Java, DTD Example 1c: Input — widl.xml.....	22-16
XML Class Generator for Java, DTD Example 1d: TestWidl.java.....	22-16
XML Class Generator for Java, DTD Example 1e: XML Output — widl.out	22-18
XML Class Generator for Java, Schema Example 1a: XML Schema, car.xsd	22-19
XML Class Generator for Java, Schema Example 1b: Application, CarDealer.java.....	22-20
XML Class Generator for Java, Schema Example 2a: Schema — book.xsd.....	22-22
XML Class Generator for Java, Schema Example 2b: BookCatalogue.java.....	22-23
XML Class Generator for Java, Schema Example 3a: Schema — po.xsd.....	22-25
XML Class Generator for Java, Schema Example 3b: Application — TestPo.java.....	22-26
Frequently Asked Questions (FAQs): Class Generator for Java.....	22-30
How Do I Install XML Class Generator?	22-30
What Does XML Class Generator for Java Do?	22-30
Which DTD's are Supported?	22-31
The Classes Not Found Error When Running XML Class Generator Samples?	22-31
In XML Class Generator, How Do I Create the Root Object More than Once?	22-31
How Can I Create XML Files from Scratch Using the DOM API?	22-32
Can I Create an XML Document in a Java Class?	22-32

Part VII XDK for Java Beans

23 Using XML Transviewer Beans

Accessing Oracle XML Transviewer Beans.....	23-2
XDK for Java: XML Transviewer Bean Features	23-2
Database Connectivity	23-2
XML Transviewer Beans.....	23-2

Using the XML Transviewer Beans	23-4
Using DOMBuilder Bean	23-5
Used for Asynchronous Parsing in the Background	23-5
DOMBuilder Bean Parses Many Files Fast	23-5
DOMBuilder Bean Usage	23-5
Using XSLTransformer Bean	23-9
Many Files to Transform? Use XSLTransformer Bean.....	23-10
Need a responsive User Interface? Use XSLTransformer Bean.....	23-10
XSL Transviewer Bean Scenario 1: Regenerating HTML — Underlying Data Changes	23-10
XSLTransformer Bean Usage	23-11
Using Treeviewer Bean	23-13
Using XMLSourceView Bean	23-15
XMLSourceView Bean Usage	23-16
Using XMLTransformPanel Bean	23-20
XMLTransformPanel Bean Features	23-20
Using DBViewer Bean	23-23
DBViewer Bean Usage	23-26
Using DBAccess Bean	23-30
DBAccess Bean Usage.....	23-30
Running the Transviewer Bean Samples	23-32
Installing the Transviewer Bean Samples	23-33
Using Database Connectivity.....	23-34
Running Makefile	23-34
Transviewer Bean Example 1: AsyncTransformSample.java.....	23-35
Transviewer Bean Example 2: ViewSample.java	23-42
Transviewer Bean Example 3: XMLTransformPanelSample.java	23-46
Transviewer Bean Example 4a: DBViewer Bean — DBViewClaims.java	23-47
Transviewer Bean Example 4b: DBViewer Bean — DBViewFrame.java	23-50
Transviewer Bean Example 4c: DBViewer Bean — DBViewSample.java.....	23-51

Part VIII XDK for C

24 Using XML Parser for C

Accessing XML Parser for C	24-2
XML Parser for C Features	24-2

Specifications	24-2
Memory Allocation.....	24-2
Thread Safety.....	24-3
Data Types Index	24-3
Error Message Files.....	24-3
Validation Modes.....	24-3
XML Parser for C Usage.....	24-4
XML Parser for C, XSLT (DOM Interface) Usage	24-6
XML Parser for C, Default Behavior	24-8
DOM and SAX APIs.....	24-9
Using the SAX API	24-9
Using the DOM API	24-10
Invoking XML Parser for C.....	24-11
Command Line Usage.....	24-11
Writing C Code to Use Supplied APIs.....	24-11
Using the Sample Files Included with Your Software	24-12
Running the XML Parser for C Sample Programs.....	24-13
Building the Sample programs	24-13
Sample Programs	24-13
XML Parser for C Example 1: XML — class.xml.....	24-13
XML Parser for C Example 2: XML — cleo.xml	24-14
XML Parser for C Example 3: XSL — iden.xsl.....	24-17
XML Parser for C Example 4: XML — FullDOM.xml (DTD).....	24-18
XML Parser for C Example 5: XML — NSExample.xml	24-18
XML Parser for C Example 6: C — DOMSample.c.....	24-19
XML Parser for C Example 7: C — DOMSample.std	24-21
XML Parser for C Example 8: C — SAXSample.c	24-21
XML Parser for C Example 9: C — SAXSample.std	24-24
XML Parser for C Example 10: C — DOMNamespace.c.....	24-25
XML Parser for C Example 11: C — DOMNamespace.std.....	24-30
XML Parser for C Example 12: C — SAXNamespace.c.....	24-30
XML Parser for C Example 13: C — SAXNamespace.std.....	24-35
XML Parser for C Example 14: C — FullDOM.c	24-36
XML Parser for C Example 15: C — FullDOM.std	24-46
XML Parser for C Example 16: C — XSLSample.c.....	24-52

XML Parser for C Example 17: C — XSLSample.std.....	24-54
---	-------

25 Using XML Schema Processor for C

Oracle XML Schema Processor for C	25-2
Oracle XML Schema for C Features	25-2
Requirements	25-2
Standards Conformance	25-3
Using the Supported Character Sets	25-3
XML Schema Processor for C: Software	25-4
Invoking XML Schema Processor for C	25-5
XML Schema Processor for C Usage Diagram	25-6
How to Run XML Schema for C Sample Programs	25-7
XML Schema for C Example 1: xsdtest.c	25-9
XML Schema for C Example 2: car.xsd.....	25-11
XML Schema for C Example 3: car.xml	25-12
XML Schema for C Example 4: car.std	25-13
XML Schema for C Example 5: aq.xsd.....	25-14
XML Schema for C Example 6: aq.xml	25-23
XML Schema for C Example 7: aq.std	25-24
XML Schema for C Example 8: pub.xsd.....	25-24
XML Schema for C Example 9: pub.xml	25-26
XML Schema for C Example 10: pub.std.....	25-27

Part IX XDK for C++

26 Using XML Parser for C++

Accessing XML Parser for C++	26-2
XML Parser for C++ Features	26-2
Specifications.....	26-2
Memory Allocation.....	26-2
Thread Safety.....	26-3
Data Types Index.....	26-3
Error Message Files	26-3
Validation Modes	26-3

XML Parser for C++ Usage	26-3
XML Parser for C++ XSLT (DOM Interface) Usage	26-6
Default Behavior	26-8
DOM and SAX APIs	26-9
Using the SAX API	26-9
Using the DOM API	26-10
Invoking XML Parser for C++	26-10
Command Line Usage.....	26-10
Writing C++ Code to Use Supplied APIs.....	26-11
Using the Sample Files Included with Your Software	26-11
Running the XML Parser for C++ Sample Programs	26-12
Building the Sample programs	26-12
Sample Programs	26-12
XML Parser for C++ Example 1: XML — class.xml	26-13
XML Parser for C++ Example 2: XML — cleo.xml	26-14
XML Parser for C++ Example 3: XSL — iden.xsl	26-16
XML Parser for C++ Example 4: XML — FullDOM.xml (DTD)	26-16
XML Parser for C++ Example 5: XML — NSExample.xml	26-16
XML Parser for C++ Example 6: C++ — DOMSample.cpp.....	26-17
XML Parser for C++ Example 7: C++ — DOMSample.std.....	26-20
XML Parser for C++ Example 8: C++ — SAXSample.cpp.....	26-22
XML Parser for C++ Example 9: C++ — SAXSample.std.....	26-25
XML Parser for C++ Example 10: C++ — DOMNamespace.cpp	26-27
XML Parser for C++ Example 11: C++ — DOMNamespace.std	26-31
XML Parser for C++ Example 12: C++ — SAXNamespace.cpp	26-31
XML Parser for C++ Example 13: C++ — SAXNamespace.std	26-35
XML Parser for C++ Example 14: C++ — FullDOM.cpp.....	26-36
XML Parser for C++ Example 15: C++ — FullDOM.std.....	26-46
XML Parser for C++ Example 16: C++ — XSLSample.cpp	26-52
XML Parser for C++ Example 17: C++ — XSLSample.std	26-54

27 Using XML Schema Processor for C++

Oracle XML Schema Processor for C++ Features	27-2
Requirements.....	27-2
Standards Conformance	27-3

Using the Supported Character Sets	27-3
XML Schema Processor for C++: Provided Software.....	27-5
Invoking XML Schema Processor for C++.....	27-5
XML Schema Processor for C++ Usage Diagram	27-6
Running the Provided XML Schema Sample Application.....	27-7
Error Messages are in English	27-8
XML Schema for C++ Example 1: xsdtest.cpp	27-8
XML Schema for C++ Example 2: car.xsd.....	27-10
XML Schema for C++ Example 3: car.xml	27-11
XML Schema for C++ Example 4: car.std	27-11
XML Schema for C++ Example 5: aq.xsd.....	27-12
XML Schema for C++ Example 6: aq.xml	27-16
XML Schema for C++ Example 7: aq.std.....	27-18
XML Schema for C++ Example 8: pub.xsd	27-18
XML Schema for C++ Example 9: pub.xml	27-20
XML Schema for C++ Example 10: pub.std.....	27-21

28 Using XML C++ Class Generator

Accessing XML C++ Class Generator	28-2
Using XML C++ Class Generator	28-2
External DTD Parsing	28-2
Error Message Files	28-2
XML C++ Class Generator Usage	28-3
xmlcg Usage	28-4
Using the XML C++ Class Generator Examples in sample/	28-5
XML C++ Class Generator Example 1: XML — Input File to Class Generator, CG.xml .	28-5
XML C++ Class Generator Example 2: DTD — Input File to Class Generator, CG.dtd ..	28-6
XML C++ Class Generator Example 3: CG Sample Program	28-6

Part X XDK for PL/SQL

29 Using XML Parser for PL/SQL

Accessing XML Parser for PL/SQL	29-2
What's Needed to Run XML Parser for PL/SQL	29-2

Using XML Parser for PL/SQL (DOM Interface)	29-2
XML Parser for PL/SQL: Default Behavior	29-5
Using the XML Parser for PL/SQL: XSL-T Processor (DOM Interface)	29-6
XML Parser for PL/SQL: XSLT Processor — Default Behavior	29-8
Using XML Parser for PL/SQL Examples in sample/	29-9
Setting Up the Environment to Run the sample/ Sample Programs.....	29-9
Running domsample	29-10
Running xlsample	29-11
XML Parser for PL/SQL Example 1: XML — family.xml.....	29-13
XML Parser for PL/SQL Example 2: DTD — family.dtd	29-13
XML Parser for PL/SQL Example 3: XSL — iden.xsl.....	29-13
XML Parser for PL/SQL Example 4: PL/SQL — domsample.sql.....	29-14
XML Parser for PL/SQL Example 5: PL/SQL — xlsample.sql.....	29-17
Frequently Asked Questions (FAQs): XML Parser for PL/SQL	29-20
Exception in Thread Parser Error	29-20
Encoding '8859_1' is not currently supported by the JVM?	29-20
xml.dom.GetNodeValue in PL/SQL	29-20
XDK for PL/SQL Toolkit.....	29-22
Parsing DTD contained in a CLOB (PL/SQL) XML.....	29-22
XML Parser for PL/SQL	29-24
Security: ORA-29532, Granting JavaSysPriv to User.....	29-24
Installing XML Parser for PL/SQL: JServer(JVM) Option.....	29-25
XML Parser for PL/SQL: domsample	29-26
XML in CLOBs	29-27
Out of memory errors in oracle.xml.parser	29-27
Is There a PL/SQL Parser Based on C?	29-29
Memory Requirements When Using the Parser for PL/SQL.....	29-29
JServer (JVM), Is It Needed to Run XML Parser for PL/SQL?	29-29
Using the DOM API	29-30
Using the Sample	29-33
XML Parser for PL/SQL: Parsing DTD in a CLOB.....	29-34
Errors When Parsing a Document.....	29-38
PLXML: Parsing a Given URL?	29-38
Using XML Parser to Parse HTML?.....	29-39
Oracle 7.3.4: Moving Data to a Web Browser (PL/SQL)	29-40

Oracle 7.3.4 and XML.....	29-40
getNodeValue(): Getting the Value of DomNode	29-41
Retrieving all Children or Grandchildren of a Node	29-41
What Causes ora-29532 "Uncaught java exception:java.lang.ClassCastException?	29-41

A An XML Primer

What is XML?	A-2
Basic Rules for XML Markup.....	A-3
W3C XML Recommendations	A-4
XML Features	A-6
How XML Differs From HTML	A-7
Presenting XML Using Stylesheets	A-10
eXtensible Stylesheet Language (XSL).....	A-10
Cascading Style Sheets (CSS)	A-11
Extensibility and Document Type Definitions (DTD)	A-11
Well-Formed and Valid XML Documents	A-12
Why Use XML?	A-13
Additional XML Resources	A-14

B Comparing Oracle XML Parsers and Class Generators by Language

Comparing the Oracle XML Parsers.....	B-2
Comparing the Oracle XML Class Generators.....	B-4

C XDK for Java: Specifications and Cheat Sheets

XML Parser for Java Cheat Sheets	C-2
Accessing XML Parser for Java	C-13
Installing XML Parser for Java, Version 2	C-13
XML Parser for Java, Version 2 Specifications	C-14
Requirements	C-15
Online Documentation.....	C-15
Release Specific Notes.....	C-15
Standards Conformance	C-15
Supported Character Set Encodings	C-16
Oracle XML Parser V1 and V2	C-17

NEW CLASS STRUCTURE	C-17
XDK for Java: XML Schema Processor	C-20
XDK for Java: XML Class Generator for Java	C-21
Installing XML Class Generator for Java.....	C-21
XML Class Generator for Java: Windows NT Installation.....	C-21
XML Class Generator for Java: UNIX Installation	C-22
XML Class Generator for Java Cheat Sheet	C-23
oracg Command Line Utility	C-25
XDK for Java: XSQL Servlet	C-26
Downloading and Installing XSQL Servlet.....	C-26
Windows NT: Starting the Web-to-go Server.....	C-27
Setting Up the Database Connection Definitions for Your Environment	C-28
UNIX: Setting Up Your Servlet Engine to Run XSQL Pages	C-28
XSQL Servlet Specifications	C-29
Character Set Support	C-29
XDK for Java: XSQL Servlet Cheat Sheets	C-30
XML SQL Utility for Java Cheat Sheet	C-32

D XDK for Java Beans: Specifications and Cheat Sheets

XDK for Javabeans: Transviewer Bean Cheat Sheet	D-2
DOMBuilder Bean Cheat Sheet	D-2
XSLTransformer Bean Cheat Sheet	D-3
XMLTreeView Bean Cheat Sheet	D-4
XMLTransformPanel Cheat Sheet	D-5
DBViewer Bean Cheat Sheet	D-6
XMLSourceView Bean Cheat Sheet	D-10
DBAccess Bean Cheat Sheet	D-14

E XDK for C: Specifications and Cheat Sheets

XML Parser for C Specifications	E-2
Validating and Non-Validating Mode Support	E-2
Example Code	E-2
Online Documentation.....	E-3
Release Specific Notes	E-3
Standards Conformance	E-3

Supported Character Set Encodings	E-3
XML Parser for C Revision History	E-5
XML Parser for C: Parser Functions	E-8
XML Parser for C: DOM API Functions	E-9
XML Parser for C: Namespace API Functions	E-12
XML Parser for C: XSLT API Functions	E-12
XML Parser for C: SAX API Functions	E-13

F XML Parser for C++: Specifications and Cheat Sheet

XML Parser for C++ Specifications	F-2
Validating and Non-Validating Mode Support	F-2
Example Code	F-2
Online Documentation	F-3
Release Specific Notes	F-3
Standards Conformance	F-3
Supported Character Set Encodings	F-3
XML Parser for C++ Revision History	F-5
XML Parser for C++: XMLParser() API	F-9
XML Parser for C++: DOM API	F-10
XML Parser for C++: XSLT API	F-15
XML Parser for C++: SAX API	F-17
XML C++ Class Generator Specifications	F-19
Input to the XML C++ Class Generator	F-19
Output to XML C++ Class Generator	F-20
Standards Conformance	F-20
Directory Structure	F-20

G XML Parser for PL/SQL: Specifications and Cheat Sheets

XML Parser for PL/SQL	G-2
Oracle XML Parser Features	G-2
Namespace Support	G-3
Validating and Non-Validating Mode Support	G-3
Example Code	G-3
IXML Parser for PL/SQL Directory Structure	G-3
DOM and SAX APIs	G-4

XML Parser for PL/SQL Specifications	G-5
XML Parser for PL/SQL: Parser() API	G-7
XML Parser for PL/SQL: XSLT Processor API	G-9
XML Parser for PL/SQL: W3C DOM API — Types	G-10
XML Parser for PL/SQL: W3C DOM API — Node Methods, Node Types, and DOM Interface Types	G-11
Node Methods	G-11
DOM Node Types	G-12
DOMException Types	G-13
DOM Interface Types	G-13

H XML SQL Utility (XSU) Specifications and Cheat Sheets

Installing XML SQL Utility	H-2
Contents of the XSU Distribution	H-2
Installing XML SQL Utility: Procedure	H-2
Installing XSU Downloaded from OTN	H-3
Requirements for Running XML SQL Utility	H-3
XSU Requirements	H-4
Extract the XSU Files	H-4
XML SQL Utility (XSU) for Java, Cheat Sheets	H-5
XML SQL Utility (XSU) for PL/SQL, Cheat Sheets	H-24
DBMS_XMLQuery PL/SQL Package	H-24
DBMS_XMLSave PL/SQL Package	H-27

Glossary

Index

Send Us Your Comments

Oracle Application Developer's Guide - XML, 10g (9.0.4)

Part No. B12099-01

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

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

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

- Electronic mail: appserverdocs_us@oracle.com
- FAX: (650) 506-7227 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

The Preface has the following sections:

- [About this Guide](#)
- [Audience](#)
- [Feature Coverage and Availability](#)
- [How this Manual is Organized](#)
- [Related Documentation](#)
- [How to Order this Manual](#)
- [Downloading Release Notes, Installation Guides, White Papers,...](#)
- [How to Access this Manual On-Line](#)
- [Conventions](#)
- [Documentation Accessibility](#)

About this Guide

This manual describes Oracle XML-enabled database technology. It describes how XML data can be stored, managed, and queried in the database using Oracle XML-enabled technology and the appropriate Oracle development tools.

After introducing you to the main criteria to consider when designing your Oracle XML application, this manual describes an overview of several scenarios that are based on real-life existing business applications. You are then introduced to the XML Developer's Kits (XDKs) and how the XDK components can work together to generate and store XML data in a database. Examples and sample applications are introduced where possible.

Examples and Sample Code

Many of the XDK examples in the manual are provided with your software in the following directories:

- `$ORACLE_HOME/xdk/java/demo/`
- `$ORACLE_HOME/xdk/C/demo/`, and so on
- `$ORACLE_HOME/xdk/java/sample/`
- `$ORACLE_HOME/rdbms/demo` directory

Composed or Decomposed (Generated) XML

In general, XML documents are processed in one of two ways:

- As *composed* XML documents, stored in LOBs
- As *decomposed* XML document fragments, stored in relational tables, with the XML tags mapped to their respective columns in the database tables. The decomposed or fragmented XML documents can then be regenerated into composed XML documents

Oracle XML-Enabled Technology

The main Oracle XML-enabled technology components are the XML Developer's Kits (XDKs). These are available in four language implementations:

- **Java.** XDK for Java, XDK for Javabeans, and XML SQL Utility for Java
- **PL/SQL.** XDK for PL/SQL and XML SQL Utility for PL/SQL
- **C.** XDK for C
- **C++.** XDK for C++

Audience

This guide is intended for developers building XML applications on Oracle Database or Oracle Application Server (OracleAS).

Prerequisite Knowledge

An understanding of XML and XSL is helpful but not essential for using this manual. References to good sources for more information are included in Appendix A and in the FAQ section of Chapter 3. An XML primer is included in Appendix A.

Many examples provided here are in either SQL, Java, PL/SQL, C, or C++, hence a working knowledge of one or more of these languages is presumed.

If you understand XML but know nothing about databases...

The best place for you to start is:

1. Read *Oracle9i Concepts*. First plan, model, and design your database.
2. Read the chapters in Part I, "[Introducing Oracle XML-Enabled Technology](#)" and Part II, "[Storing and Retrieving XML From the Database](#)".
3. Visit Oracle Technology Network (OTN) sites at:
 - <http://otn.oracle.com> for general information about database features
 - <http://otn.oracle.com/tech/xml> for information about the XML Developer's Kits (XDKs) available as well as white papers and demos.
4. Check the Frequently Asked Questions (FAQ) sections in this manual, starting with those at the end of:
 - [Chapter 3 — "Frequently Asked Questions \(FAQs\): Oracle XML-Enabled Technology"](#) on page 3-26
 - [Chapter 7 — "Frequently Asked Questions \(FAQs\): XML SQL Utility \(XSU\)"](#) on page 7-2
 - [Chapter 8 — "Frequently Asked Questions \(FAQs\): Oracle Text"](#) on page 8-50
 - [Chapter 10 — "Frequently Asked Questions \(FAQs\) - XSQL Servlet"](#) on page 10-74
 - [Chapter 20 — "Frequently Asked Questions \(FAQs\): XML Parser for Java"](#) on page 20-55

5. If you still have questions, consult with your Oracle representative or, to help get you started, go to the “Discussions” option on OTN and post your question there.
6. Of course, once you have determined which language you need for your application and which XDK components you need to build your application, for detail on the XML components and how they are used, see:
 - [Chapter 7, Chapter 10, and Part IV — Tools and Frameworks for Building Oracle-Based XML Applications](#) through [Part X — XDK for PL/SQL](#), of this manual.
 - *Oracle9i XML Reference*

If you understand databases but know nothing about XML...

The best place for you to start is:

1. Read [Appendix A, "An XML Primer"](#) and the references at the end of [Chapter 3, "Oracle XML Developer Kits \(XDKs\) and Components: Overview and General FAQs"](#). There are many good books and web sites that introduce you to XML. Some of these are listed in Appendix A.
2. Read [Chapter 4, "Using XSL and XSLT"](#).
3. Read the FAQs at the end of the chapters, starting with:
 - [Chapter 3 — "Frequently Asked Questions \(FAQs\): Oracle XML-Enabled Technology"](#) on page 3-26
 - [Chapter 7 — "Frequently Asked Questions \(FAQs\): XML SQL Utility \(XSU\)"](#) on page 7-2
 - [Chapter 8 — "Frequently Asked Questions \(FAQs\): Oracle Text"](#) on page 8-50
 - [Chapter 10 — "Frequently Asked Questions \(FAQs\) - XSQL Servlet"](#) on page 10-74
 - [Chapter 20 — "Frequently Asked Questions \(FAQs\): XML Parser for Java"](#) on page 20-55
4. Of course you need to read about the (new) native XML support in Oracle Database, so read the chapters in Part I, ["Introducing Oracle XML-Enabled Technology"](#) and Part II, ["Storing and Retrieving XML From the Database"](#).

5. Visit the Oracle Technology Network (OTN) XML site at <http://otn.oracle.com/tech/xml> for information about the XML Developer Kits (XDKs) available as well as white papers and demos.
6. Of course, once you have determined which language you need for your application and which XDK components you need to build your application, for detail on the XML components and how they are used, see:
 - [Chapter 7](#), [Chapter 10](#), and Part IV — [Tools and Frameworks for Building Oracle-Based XML Applications](#) through Part X — [XDK for PL/SQL](#), of this manual.
 - *Oracle9i XML Reference*
7. If you still have questions, consult with your Oracle representative or, to help get you started, go to the “Discussions” option on OTN and post your question there.

Feature Coverage and Availability

Information in this manual represents a snapshot of information on Oracle XML-enabled technology components. These change rapidly. To view the latest information, refer to Oracle Technology Network (OTN) at: <http://otn.oracle.com/tech/xml>

How this Manual is Organized

This manual is organized into 10 parts, 29 chapters, and 8 appendixes. It includes an index and glossary.

Roadmap of this Manual

A detailed version of this diagram is provided in [Chapter 1, "Oracle XML-Enabled Technology"](#).

- ***Introducing XML and the Oracle Database.*** Introductory and basic information about using Oracle Database XML components (Chapters 1 through 8), XML support in the database, using XMLType and URI-Reference, XML SQL Utility (XSU), and how to apply Oracle Text to search and retrieve information from XML documents.
 - PART 1 "[Introducing Oracle XML-Enabled Technology](#)"
 - * [Chapter 1, "Oracle XML-Enabled Technology"](#), introduces you to Oracle XML Developer Kits (XDKs), the (new) native XML support in the

database and XMLType and DBUri-ref, tools used to build XML applications, and Oracle Text (*interMedia* Text).

- * [Chapter 2, "Modeling and Design Issues for Oracle XML Applications"](#), describes some XML design and loading issues, and how Oracle XML components can be used in typical content/document management and business-to-business messaging applications.
- * [Chapter 3, "Oracle XML Developer Kits \(XDKs\) and Components: Overview and General FAQs"](#), introduces you to the Oracle XML components, the XML Development Kits and XML SQL Utility. It summarizes the ways you can generate XML documents for each language, Java, C, C++, and PL/SQL. It provides Frequently Asked Questions (FAQs) about Oracle's XML-enabled technology.
- * [Chapter 4, "Using XSL and XSLT"](#), introduces you to XML and XSLT. It also discusses the differences between Cascading Style Sheets (CSS) and XSL. This chapter includes Frequently Asked Questions.
- **PART II "Storing and Retrieving XML From the Database"**
 - * [Chapter 5, "Database Support for XML"](#), introduces the (new) datatype XMLType and describes how to use XMLType when generating and storing XML in the database. This chapter also describes the `extract()` and `existsnode()` functions and member functions, DBMS_XMLGEN package, SYS_XMLGEN function used to generate XML in SQL queries, SYS_XMLAGG function used to aggregate XML data, creating and managing XMLTypes, and using functional indexes and table functions to query XML.
 - * [Chapter 6, "Database Uri-references"](#), describes Uri-Reference (Uri-ref), and DBUri-ref, how to use URIType and subtypes, DBUriType and HttpUriType, the new SYS_DBURIGEN() operator, URIFactory Method, HTTP access to objects and the OraDBUriServlet mechanism.
 - * [Chapter 7, "XML SQL Utility \(XSU\)"](#), describes how to use XML SQL Utility Java and PL/SQL APIs, to generate and 'store' XML documents, how to insert, update, and delete XML documents in the database, use the XSU command line tool, and map elements to columns. Examples in this chapter are also available from `$ORACLE_HOME/rdbms/demo/xsu`. This chapter also provides Frequently Asked Questions (FAQs).

- * [Chapter 8, "Searching XML Data with Oracle Text"](#), introduces you to Oracle Text (*interMedia* Text), using the CONTAINS operator, how to create an Oracle Text section and index, and how to build queries. It includes examples and guidelines on using XML_SECTION_GROUP, AUTO_SECTION_GROUP, the new PATH_SECTION_GROUP, the INPATH and HASPATH operators. This chapter also provides Frequently Asked Questions (FAQs).
- ***B2B and XML Data Exchange.***
 - Part III. ["Data Exchange Using XML"](#)
 - * [Chapter 9, "Exchanging XML Data Using Oracle AQ"](#), introduces you to some Advanced Queueing (AQ) concepts and describes how AQ and XML complement each other. This chapter also describes the (new) Internet-Data-Access-Presentation (IDAP) mechanism, the AQXMLServlet and accessing it with HTTP and SMTP. It also describes XMLType Queues and XML AQ message transformation. This chapter provides several FAQs.
 - ***Oracle Development Tools.*** Chapters 10 through 17 describes either introductory information or how to use XSQL Pages Publishing Framework, JDeveloper, Business Components for Java (BC4J), the (new) Metadata API, Oracle Reports, Oracle Portal, Oracle Exchange, and XML Gateway.
 - PART IV ["Tools and Frameworks for Building Oracle-Based XML Applications"](#)
 - * [Chapter 10, "XSQL Pages Publishing Framework"](#), provides some insight on using XSQL Servlet. It includes diagrams that explain how the XSQL Page Processor works. This chapter includes FAQs.
 - * [Chapter 11, "Using JDeveloper to Build Oracle XML Applications"](#), introduces you using JDeveloper for building XML applications, using XSQL servlet from JDeveloper, and steps to take when about building a Mobile application with JDeveloper. This chapter includes FAQs.
 - * [Chapter 12, "Building BC4J and XML Applications"](#), introduces you to Business Components for Java (BC4J) and how to use the BC4J framework to build XML applications.
 - * [Chapter 13, "Using Metadata API"](#), describes the Metadata API and how to use it. It includes a description of the (new) DBMS_METADATA's programmatic and browsing interfaces, as well as a detailed example.

- * [Chapter 14, "OracleAS Reports Services and XML"](#), describes how to generate and customize reports Reports as XML and read XML data source from reports. It includes information about using (new) pluggable XML data sources, XML-PDS, and their support for XSQL.
- * [Chapter 15, "Using the PDK for Visualizing XML Data in Oracle Portal"](#), briefly describes the Oracle Application Server Portal features, Portal Developer Kit (PDK) and how you can use URL Services to enable you use your XML application as a web or database-based portlet.
- * [Chapter 16, "How Oracle Exchange Uses XML"](#) introduces you to Oracle Exchange, its stored and pass through transactions, XML delivery formats, and e-business solution architecture.
- * [Chapter 17, "Introducing Oracle XML Gateway"](#) -- introduces you to Oracle XML Gateway. Oracle XML Gateway is a set of services that allows for easy integration with the Oracle e-Business Suite to create and consume XML messages triggered by business events.
- **PART V "OracleAS Dynamic Services (DS) and Oracle Syndication Server (OSS)"**
 - * [Chapter 18, "Using OracleAS Dynamic Services and XML"](#), introduces you to OracleAS Dynamic Services, its architecture, Java, PL/SQL, and JMS/HTTP deployment modes, and developing Dynamic Services services.
 - * [Chapter 19, "Oracle Syndication Server \(OSS\) and XML"](#), provides an overview of Oracle Syndication Server (OSS), its use of the ICE protocol, OSS architecture, and how OSS interfaces with Oracle Dynamic Services, content providers, and subscribers.
- **XML Developer Kits (XDKs).** The roadmap shows the various XML Developer Kits (XDKs), in the "XML Application" box. Chapter 7, and 19 through 29 describe how to use the XDKs.
 - **Part VI "XDK for Java"**
 - * [Chapter 20, "Using XML Parser for Java"](#), describes ways of using XML Parser for Java and XSLT Processor. It lists the examples provided with the software. This chapter includes FAQs.
 - * [Chapter 21, "Using XML Schema Processor for Java"](#), introduces you to XML Schema, compares XML Schema to DTDs, and describes Oracle XML Schema Processor Java features, usage, how to use the sample program. It also provides FAQs.

- * [Chapter 22, "XML Class Generator for Java"](#), describes ways of using XML Java Class Generator with DTDs and XML Schema. It lists the examples provided with the software. This chapter includes FAQs.
- Part VII ["XDK for Java Beans"](#)
 - * [Chapter 23, "Using XML Transviewer Beans"](#), discusses the XML Transviewer Beans and how to use them, including the (new) DBViewer bean and (new) DBAccess bean. It lists examples provided with your software.
- Part VIII ["XDK for C"](#)
 - * [Chapter 24, "Using XML Parser for C"](#), describes ways of using XML Parser for C and XSLT Processor. It lists the examples provided with the software.
 - * [Chapter 25, "Using XML Schema Processor for C"](#), describes the XML Schema Process for C features, calling sequence, and how to run the supplied sample programs. This chapter also lists the supplied examples.
- Part IX ["XDK for C++"](#)
 - * [Chapter 26, "Using XML Parser for C++"](#), describes ways of using XML Parser for C++ and XSLT Processor. It lists the examples provided with the software.
 - * [Chapter 27, "Using XML Schema Processor for C++"](#), describes the XML Schema Process for C++ features, calling sequence, and how to run the supplied sample programs. This chapter also lists the supplied example.
 - * [Chapter 28, "Using XML C++ Class Generator"](#), describes ways of using XML C++ Class Generator. It lists the examples provided with the software.
- Part X ["XDK for PL/SQL"](#)
 - * [Chapter 29, "Using XML Parser for PL/SQL"](#), describes ways of using XML Parser for PL/SQL and XSLT Processor. It lists the examples provided with the software. This chapter includes FAQs.

Not shown in the roadmap are the Appendixes which include the XML Primer, and XDK cheat sheets and specifications.

- [Appendix A, "An XML Primer"](#), introduces you to some basic and background information about XML.

- [Appendix B, "Comparing Oracle XML Parsers and Class Generators by Language"](#), compares the Oracle XML Parsers and Class Generators according to implementation language.
- [Appendix C, "XDK for Java: Specifications and Cheat Sheets"](#), describes the XDK for Java component specifications. Includes several top level class and method listings.
- [Appendix D, "XDK for Java Beans: Specifications and Cheat Sheets"](#), describes the XDK for Java Beans, specifically the Transviewer Beans cheatsheets.
- [Appendix E, "XDK for C: Specifications and Cheat Sheets"](#), describes the XDK for C specifications. Includes top level function listings.
- [Appendix F, "XDK for C++: Specifications and Cheat Sheet"](#), describes the XDK for C++ component specifications. Includes several top level class and method listings.
- [Appendix G, "XDK for PL/SQL: Specifications and Cheat Sheets"](#), describes the XDK for PL/SQL specifications. Includes several top level function listings.
- [Appendix H, "XML SQL Utility \(XSU\) Specifications and Cheat Sheets"](#), describes the XML SQL Utility (XSU) for Java and PL/SQL specifications. Includes several top level method and function listings.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i New Features* for information about the differences between Oracle9i and the Oracle9i Enterprise Edition and the available features and options. That book also describes all the features that are new in Oracle9i.
- *Oracle9i Concepts*.
- *The JDeveloper Guide*
- *Oracle9i Application Developer's Guide - Fundamentals*
- *Oracle8i Application Developer's Guide - Advanced Queuing*
- *Oracle9i Supplied PL/SQL Packages Reference*
- *Oracle Integration Server Overview*
- *Oracle9i XML Reference*
- *The Oracle XML Handbook*, XML Core Development Team, Oracle., Oracle Press

- *Building Oracle XML Applications*, Steve Muench, O'Reilly
- *XML Bible*, Elliotte Rusty Harold, IDG Books Worldwide
- *XML Unleashed*, Morrison et al., SAMS
- *Building XML Applications*, St. Laurent and Cerami, McGraw-Hill
- *Building Web Sites with XML*, Michael Floyd, Prentice Hall PTR
- *Building Corporate Portals with XML*, Finkelstein and Aiken, McGraw-Hill
- *XML in a Nutshell*, O'Reilly
- *Learning XML - (Guide to) Creating Self-Describing Data*, Ray, O'Reilly
- <http://www.xml.com/pub/rg/46>
- <http://www.xmlmag.com/>
- <http://www.webmethods.com/>
- <http://www.clarient.org/>
- <http://www.xmlwriter.com/>
- http://webdevelopersjournal.com/articles/why_xml.html
- <http://www.w3schools.com/xml/>
- <http://www.w3.org/TR/REC-xml>

How to Order this Manual

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/membership/index.htm>

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>

Downloading Release Notes, Installation Guides, White Papers,...

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

<http://otn.oracle.com/membership/index.htm>

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>

How to Access this Manual On-Line

You can find copies of or download this manual from any of the following locations:

- On the Document CD that accompanies your Oracle Database software CD
- From Oracle Technology Network (OTN) at <http://otn.oracle.com/docs/index.htm>, under Data Server (or whatever other product you have). For example, select Oracle9i > General Documentation Release 1 (9.0.1) (or whatever other section you need to specify). Select HTML then select HTML or PDF for your particular of interest, such as, “Oracle Documentation Library”. Note that you may only be able to locate the prior release manuals at this site.

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 .
<i>Italics</i>	Italic typeface indicates book titles or emphasis.	<i>Oracle9i Concepts</i> Ensure that the recovery catalog and target database do <i>not</i> 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 monospace (fixed-width font) <i>italic</i>	Lowercase monospace italic font represents placeholders or variables.	You can specify the <i>parallel_clause</i> . Run <i>Uold_release</i> .SQL where <i>old_release</i> 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 monospaced (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 (<i>digits</i> [, <i>precision</i>])
{ }	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: <ul style="list-style-type: none"> That we have omitted parts of the code that are not directly related to the example That you can repeat a portion of the code 	CREATE TABLE ... AS <i>subquery</i> ; SELECT <i>col1</i> , <i>col2</i> , ... , <i>coln</i> 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;
<i>Italics</i>	Italicized text indicates placeholders or variables for which you must supply particular values.	CONNECT SYSTEM/ <i>system_password</i> DB_NAME = <i>database_name</i>
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;

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

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 in Oracle XML-Enabled Technology?

This section describes the new features in the following releases:

- **Features Introduced with Oracle Application Developer's Guide - XML, Release 10g (9.0.4)**

Features Introduced with Oracle Application Developer's Guide - XML, Release 10g (9.0.4)

Here are the new XML features in Release 10g (9.0.4):

XDK for Java

- XML Schema Processor for Java
- XML Parser for Java — DOM 2.0 and SAX 2.0 support
- Improved XSLT performance

See:

- [Chapter 20, "Using XML Parser for Java"](#)
- [Chapter 21, "Using XML Schema Processor for Java"](#)

- Class Generator for Java now includes XML Schema based Class Generator as well as a DTD based Class Generator

See: [Chapter 22, "XML Class Generator for Java"](#)

■ XSQL Servlet and Pages

- Support for Database Bind Variables. Now both lexical substitution and true database bind variables are supported for improved performance.
- Support for PDF Output Using Apache FOP. You can now combine XSQL Pages with the Apache FOP processor to produce Adobe PDF output from any XML content.
- Trusted Host Support for XSLT Stylesheets. New security features insure that stylesheets cannot be executed from non-trusted hosts.
- Full Support for Non-Oracle JDBC Drivers. Now all query, insert, update, and delete features with both Oracle and Non-Oracle JDBC drivers.
- Process Dynamically Constructed XSQL Pages. The XSQLRequest API can now process programmatically constructed XSQL pages.
- Use a Custom Connection Manager. You can now implement your own Connection Manager to handle database connections in any way you like.
- Produce Inline XML Schema. You can now optionally produce an inline XML Schema that describes the structure of your XML query results.

- Set Default Date Format for Queries. You can now supply a date format mask to change the default way date data is formatted.
- Write Custom Serializers. You can create and use custom serializers that control what and how the XSQL page processor will return to the client.
- Dynamic Stylesheet Assignment. Assign stylesheets dynamically based on parameters or the result of a SQL query.
- Update or Delete Posted XML. In addition to inserting XML, now updating and deleting is also supported.
- Insert or Update Only Targeted Columns. You can now explicitly list what columns should be included in any insert or update request.
- Page-Request Scoped Objects. Your action handlers can now get/set objects in the page request context to share state between actions within a page.
- Access to ServletContext. In addition to accessing the HttpRequest and HttpResponse objects, you can also access the ServletContext.

See: [Chapter 10, "XSQL Pages Publishing Framework"](#)

- **XDK for Java Beans**

- DBViewer Bean (new). Displays database queries or any XML by applying XSL stylesheets and visualizing the resulting HTML in a scrollable swing panel.
- DBAccess Bean (new). DB Access bean maintains CLOB tables that hold multiple XML and text documents.

See: [Chapter 23, "Using XML Transviewer Beans"](#)

- **XDK for C**

- XML Parser for C — DOM 1.0 plus DOM CORE 2.0 (a subset of DOM)
- XML Schema Processor for C
- Improved XSLT performance

See: [Chapter 25, "Using XML Schema Processor for C"](#)

- **XDK for C++**

- XML Parser for C++ — DOM 1.0 plus DOM CORE 2.0 (a subset of DOM)

- XML Schema Processor for C++
- Improved XSLT performance
 - See:** [Chapter 27, "Using XML Schema Processor for C++"](#)
- **XDK for PL/SQL**
 - Improved XSLT performance
 - See:** [Chapter 29, "Using XML Parser for PL/SQL"](#)

XML SQL Utility (XSU) Features

- Ability to generate XML Schema given an SQL Query
- Support for XMLType and Uri-ref
- Ability to generate XML as a stream of SAX2 callbacks
- XML attribute support when generation XML from the database. This provides an easy way of specifying that a particular column or group of columns should be mapped to an XML attribute instead of an XML element.

XSU is also considered part of the XDK for Java and XDK for PL/SQL.

See: [Chapter 7, "XML SQL Utility \(XSU\)"](#)

Database XML Related Enhancements

Extensible Markup Language (XML) is a standard format developed by the World Wide Web Consortium (W3C) for representing structured and unstructured data on the Web. Universal Resource Identifiers (URIs) identify resources such as Web pages anywhere on the Web. Oracle provides types to handle XML and URI data, as well as a class of URIs called `DBUri-REFs` to access data stored within the database itself. It also provides a new set of types to store and access both external and internal URIs from within the database.

XMLType

This (new) Oracle-supplied type can be used to store and query XML data in the database. `XMLType` has member functions you can use to access, extract, and query the XML data using XPath expressions. XPath is another standard developed by the W3C committee to traverse XML documents. Oracle `XMLType` functions support a subset of the W3C XPath expressions. Oracle also provides a set of SQL functions (including `SYS_XMLGEN` and `SYS_XMLAGG`) and PL/SQL packages (including

DBMS_XMLGEN) to create XMLType values from existing relational or object relational data.

XMLType is a system-defined type, so you can use it as an argument of a function or as the datatype of a table or view column. When you create a XMLType column in a table, Oracle internally uses a CLOB to store the actual XML data associated with this column. As is true for all CLOB data, you can make updates only to the entire XML document. You can create an Oracle Text index or other function-based index on a XMLType column.

URI Datatypes

Oracle supplies a family of URI types—UriType, DBUriType, and HttpUriType—which are related by an inheritance hierarchy. UriType is an object type and the others are subtypes of UriType.

- You can use HttpUriType to store URLs to external web pages or to files. It accesses these files using HTTP (Hypertext Transfer Protocol).
- DBUriType can be used to store DBUri-REFs, which reference data inside the database. Since UriType is the supertype, you can create columns of this type and store DBUriType or HttpUriType type instances in this column. Doing so lets you reference data stored inside or outside the database and access the data consistently.

DBUri-REFs use an XPath-like representation to reference data inside the database. If you imagine the database as a XML tree, then you would see the tables, rows, and columns as elements in the XML document. For instance, the sample human resources user hr would see the following XML tree:

```
<HR>
  <EMPLOYEES>
    <ROW>
      <EMPLOYEE_ID>205</EMPLOYEE_ID>
      <LAST_NAME>Higgins</LAST_NAME>
      <SALARY>12000</SALARY>
      .. <!-- other columns -->
    </ROW>
    ... <!-- other rows -->
  </EMPLOYEES>
  <!-- other tables.-->
</HR>
<!-- other user schemas on which you have some privilege on.-->
```

The `DBUri-REF` is simply an XPath expression over this virtual XML document. So to reference the `SALARY` value in the `EMPLOYEES` table for the employee with employee number 205, we can write a `DBUri-REF` as,

```
/HR/EMPLOYEES/ROW[EMPLOYEE_ID=205]/SALARY
```

Using this model, you can reference data stored in `CLOB` columns or other columns and expose them as URLs to the external world. Oracle provides a standard URI servlet that can interpret such URLs. You can install and run this servlet under the Oracle Servlet engine.

UriFactoryType

`UriFactoryType` is a **factory type**, which is a type that can create and return other object types. When given a URL string, `UriFactoryType` can create instances of the various subtypes of the `UriTypes`. It analyzes the URL string, identifies the type of URL (`HTTP`, `DBUri`, and so on) and creates an instance of the subtype.

See:

- [Chapter 5, "Database Support for XML"](#)
- [Chapter 6, "Database Uri-references"](#)
- [Chapter 8, "Searching XML Data with Oracle Text"](#)

Advanced Queueing (AQ) Features

New Advanced Queueing features include enhanced XML messaging options:

- Internet-Data-Access-Presentation (IDAP)
- AQXMLServlet for use with HTTP and SMTP access
- XMLType Queues
- XML AQ message transformation

See: [Chapter 9, "Exchanging XML Data Using Oracle AQ"](#)

Metadata API

Metadata API (new) provides a centralized, simple and flexible means for performing the following tasks:

- Extracting complete definitions of database objects (metadata) as either XML or creation DDL

- Transforming metadata via industry-standard XSLT (XML Stylesheet Transformation language).
- Generating SQL DDL to recreate the database objects

Metadata API is available on Oracle Database whenever the instance is operational. It is not available on Oracle Lite. It includes the (new) DBMS_METADATA PL/SQL supplied package.

See: [Chapter 13, "Using Metadata API"](#)

Oracle Text (*interMedia* Text/Context) Features

The new Oracle Text section group, PATH_SECTION_GROUP, enables new and more sophisticated section searching for XML documents. PATH_SECTION_GROUP supports the following:

- Case sensitivity
- Searching multi-tag paths with direct parentage ensured
- Path searching to wildcard levels
- Searching to reference top-level tags
- Attribute value sensitive searching, searching by section existence

The new Oracle Text operators are:

- * HASPATH() operator
- * INPATH() operator

See: [Chapter 8, "Searching XML Data with Oracle Text"](#)

OracleAS Reports Services

- Server enhancements: For Java based servers — Re-engineered in 100% Java. Status information is now available in HTML and XML.
- Pluggable Datasources and Destinations (new). Create your own data-access-modules in JAVA.
 - * Plug into Reports using PDS-API
 - * Integrate seamlessly into data model
 - * combine multiple data sources in a single report
 - * Shipped PDSs (XML, JDBC, Express, and SQL)

- JSP Based runtime
- Enhanced portal integration, report bursting, e-mail, distribution, and PDF support
- Event based reporting

See Also: [Chapter 14, "OracleAS Reports Services and XML"](#)

Part I

Introducing Oracle XML-Enabled Technology

Part I of the book introduces you to Oracle XML-enabled technology and features, Oracle XML Developer's Kits (XDKs) and XML components, modeling and design issues, and example scenarios. Chapter 4 provides some basic information about using XSL and XSLT.

Part I contains the following chapters:

- [Chapter 1, "Oracle XML-Enabled Technology"](#)
- [Chapter 2, "Modeling and Design Issues for Oracle XML Applications"](#)
- [Chapter 3, "Oracle XML Developer Kits \(XDKs\) and Components: Overview and General FAQs"](#)
- [Chapter 4, "Using XSL and XSLT"](#)

Oracle XML-Enabled Technology

This chapter describes the following sections:

- [What is XML?](#)
- [Storing and Retrieving XML Data from Oracle](#)
- [XML Support in the Database](#)
- [Oracle-Based XML Applications](#)
- [Oracle XML-Enabled Technology Components and Features](#)
- [The Oracle Suite of Integrated Tools and Components](#)
- [Oracle XML Samples and Demos](#)
- [What Is Needed to Run Oracle XML Components](#)
- [XML Technical Support](#)

What is XML?

[Appendix A, "An XML Primer"](#), provides some introductory information about XML, the W3C XML recommendations, differences between HTML and XML, and other XML syntax topics. It also discusses reasons why XML, the internet standard for information exchange is such an appropriate and necessary language to use in database applications.

What are Oracle XML-Enabled Technologies?

XML models structured and semi-structured data. Oracle supports structured and semi-structured data, as well as complex and unstructured data. Oracle Database is XML-enabled in that it natively handles the storage, query, presentation, and manipulation of XML data.

Oracle XML Components

[Figure 1-1](#) shows the Oracle XML components in the "XML application" box. Oracle XML components are comprised of the following:

- Database XML support
 - `XMLType` - a new datatype to store, query, and retrieve XML documents
 - `SYS_XMLGEN` - SQL function to create XML documents
 - `SYS_XMLAGG` - SQL function to aggregate multiple XML documents
 - `DBMS_XMLGEN` - a built-in package to create XML from SQL queries
 - URI support - store and retrieve global and intra-database references
 - Text support - Supports XPath on `XMLType` and text columns
- XML Developer's Kit (XDK) for Java
 - XML Parser for Java and XSLT Processor
 - XML Schema Processor for Java
 - XML Class Generator for Java
 - XSQL Servlet
 - XML SQL Utility (XSU) for Java
- XDK for Java Beans
 - XML Transviewer Beans

- * DOMBuilder Bean
- * XSLTransformer Bean
- * DBAccessBean
- * TreeViewer Bean
- * SourceViewer Bean
- * XMLTransformPanel Bean
- * DBViewer Bean
- XDK for C
 - XML Parser for C
 - XML Schema Processor for C
- XDK for C++
 - XML Parser for C++
 - XML Schema Processor for C++
 - XML Class Generator for C++
- XDK for PL/SQL
 - XML Parser for PL/SQL
 - XML SQL Utility (XSU) for PL/SQL

Figure 1–1 also lists some typical XML-based business solutions:

- Business Data Exchanges with XML
 - Buyer-Supplier Transparent Trading Automation
 - Seamless integration of partners and HTTP-based data exchange
 - Database inventory access and integration of commercial transactions and flow
 - Self-service procurement, such as using Oracle iProcurement
 - Data mining and reporting with Oracle Discoverer 3i Viewer
 - Oracle Exchange and Applications
 - Phone number portability
- Content and Document Management with XML

- Personalized publishing and portals
- Customized presentation. Dynamic News case study, Portal-to-Go, and Flight Finder

Figure 1-1 Oracle XML Components and E-Business Solutions: What's Involved

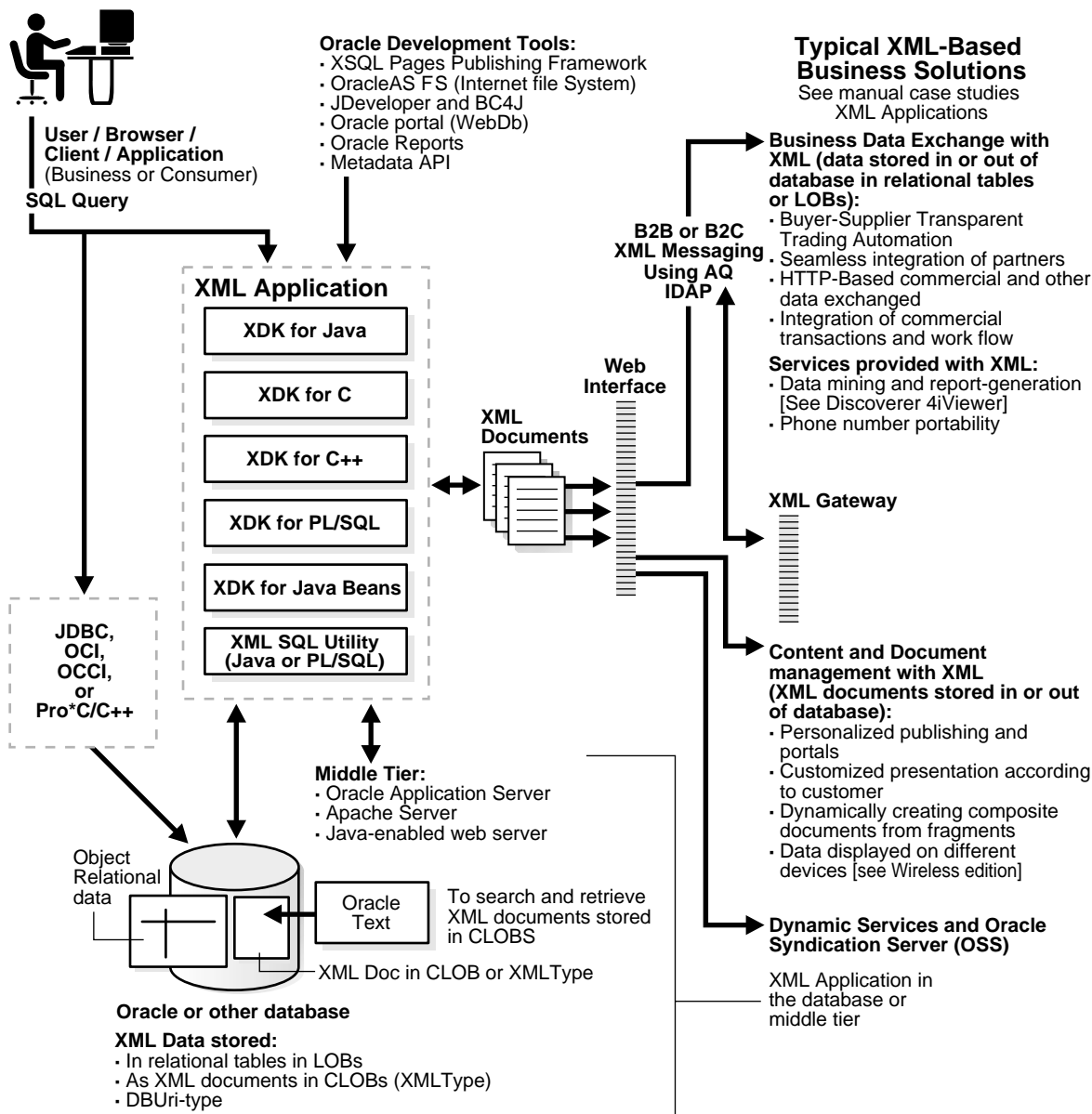


Figure 1-1 also shows the following:

Oracle Development Tools and Frameworks

XSQL Servlet and Pages, Oracle Internet File System (iFS), JDeveloper, Business Components for Java (BC4J), Oracle Portal (WebDB), Oracle Application Server Reports Services, and Oracle Dynamic Services can all be used to build XML applications.

Note: XSQL Servlet and Pages are part of the Oracle XDK for Java.

Database and Middle Tier

XML applications, such as Oracle Application Server, Apache Server, or other Java-enabled Web servers can either reside on the database or on a middle tier.

Data Stored in the Database

Data is stored as relational tables utilizing object views or as XML documents in `XMLType` columns and CLOBs. Oracle Text (*interMedia Text*) can be used to efficiently search XML documents stored in `XMLType` or CLOB columns.

Storing and Retrieving XML Data from Oracle

XML has emerged as the standard for data interchange on the Web and Oracle is XML-enabled to natively store, search, and retrieve XML in the following formats:

- **As decomposed XML documents.** That is, when the XML documents are stored in their constituent fragments. Here the XML data is stored in object relational form and you can use XML SQL Utility (XSU) or SQL functions and packages to generate the whole (composed) XML documents from these object relational instances.

You can also use XSU or SQL functions, such as `Extract()`, and `TABLE` functions, to convert the XML back to its object relational (decomposed) form.

- **As composed, or "whole" XML documents.** Store XML data in `XMLType` or CLOB/BLOB columns and use `XMLType` functions such as `Extract()` and `ExistsNode()` or Oracle Text indexing to search these documents.

XML Support in the Database

Oracle Database provides the following XML support:

- **Generation of XML Documents.** Oracle supports the generation of XML on the client or server, where existing object-relational data can be used to generate the corresponding XML. XML can be generated from query results or as part of the SQL query itself using the XSU (XML SQL Utility) Java or PL/SQL API.

In this release, Oracle extends the XML support in the server:

- By providing new SQL functions for XML generation and aggregation
- By providing a C version of the XML SQL Utility, linked to the server
- **Storage, Querying, and Retrieval of XML documents.** Before this release you could use, for example, XSU to store, query, and retrieve XML documents. Now, with this release you can use the new datatype, XMLType.

XMLType stores XML documents as Character Large Objects (CLOBs). Oracle Text (*interMedia* Text) indexing can then be used to index the XMLType columns and query them using the CONTAINS operator and an XPath-like syntax. XMLType also supports member functions that can be used to extract fragments from the XML document.

See Also: [Chapter 5, "Database Support for XML", "Indexing XMLType columns"](#) on page 5-30.

XML and URI Data Types

Oracle provides new types to handle XML and URI data. The Extensible Markup Language (XML) is a standard format developed by the World Wide Web Consortium (W3C) for representing structured and un-structured data on the Web.

URIs or Universal Resource Identifiers are used to identify resources such as web pages anywhere on the web. Oracle provides a new class of URIs to access data stored in the database itself, called *DBUri-refs*. It also provides a new set of types to store and access both external and internal URIs from the database.

XMLType

The Oracle supplied type, XMLType, can be used to store and query XML data in the database. XMLType provides member functions to access, extract and query the XML data using XPath expressions. XPath is another standard developed by the W3C committee to traverse XML documents. XMLType functions only support a limited subset of the XPath expressions. Oracle also provides a set of SQL functions

such as `SYS_XMLGEN`, `SYS_XMLAGG`, and other PL/SQL packages (`DBMS_XMLGEN`) to create these `XMLType` values from existing relational or object relational data.

`XMLType`, a system defined type, can be used as arguments to functions or as table or view columns. When you create a `XMLType` column in a table Oracle internally uses a CLOB to actually store the XML data associated with this column. You can create Oracle Text indexing on the `XMLType` column and other functional indexes. In Oracle Database, since the `XMLType` is stored as a CLOB, updates can only be made to the entire document.

See Also:

- [Chapter 5, "Database Support for XML"](#)
- [Chapter 8, "Searching XML Data with Oracle Text"](#)
- [Chapter 9, "Exchanging XML Data Using Oracle AQ"](#)
- *Oracle9i Application Developer's Guide - Advanced Queuing*, or information about using `XMLType` with Oracle Advanced Queuing

URI Data Types

Oracle supplies the following family of Uri types:

- `UriType`
- `DBUriType`
- `HttpUriType`

These are related by an inheritance hierarchy. `UriType` is an abstract type and the `DBUriType` and `HttpUriType` are subtypes of this type.

- `HttpUriType` can be used to store URLs to external web pages or files. It accesses these files using the HTTP protocol (Hyper Text Transfer Protocol).
- `DBUriType` can be used to store `DBUri`-refs which reference data inside the database.

Since `UriType` is the super type, you can create columns of this type and store `DBUriType` or `HttpUriType` instances in this column. This allows you to reference data stored inside or outside the database and access them consistently.

`DBUri`-ref uses an XPath like representation to reference data inside the database. If you imagine the database as a XML tree, then you would see the tables, rows and

columns as elements in the XML document. For instance, user `scott` would see a tree such as:

```
<SCOTT>
  <EMP>
    <ROW>
      <EMPNO>2100</EMPNO>
      <ENAME>John</ENAME>
      <SALARY>10000</SALARY>
      .. <!-- other columns -->
    </ROW>
    ... <!-- other rows -->
  </EMP>
  <!-- other tables.-->
</SCOTT>
<!-- other user schemas on which you have some privilege on.-->
```

DBUri-ref is simply an XPath expression over this virtual XML document. So to reference the `SALARY` value in the `EMP` table for the employee with employee number 2100, you can write a DBUri-ref as:

```
/SCOTT/EMP/ROW[EMPNO=2100]/SALARY
```

Using this, you can reference data stored in CLOBs or other columns and expose them as URLs to the external world. Oracle provides a standard servlet that can be installed and run under the Oracle Servlet engine which can interpret such URLs.

See Also: [Chapter 6, "Database Uri-references"](#)

Extensibility and XML

Oracle's extensibility enables special indexing on XML, including Oracle Text indexes for section searching, special operators to process XML, aggregation of XML, and special optimization of queries involving XML.

Oracle Text Searching

XML text stored in LOBs can be indexed using the extensibility indexing interface. Oracle provides operators such as `CONTAINS` and `WITHIN` that you can use to search within the XML text for substring matches.

See Also:

- [Chapter 2, "Modeling and Design Issues for Oracle XML Applications"](#).
- [Chapter 8, "Searching XML Data with Oracle Text"](#)

Oracle-Based XML Applications

There are many potential uses of XML in Internet applications. This manual focuses on the following two database-centric application areas where Oracle's XML components are well suited.

Content and Document Management

Content and document management includes customizing data presentation. These applications typically process mostly authored XML documents. Several case studies are described in the manual.

See:

- ["Customizing Content with XML: Dynamic News Application"](#)
- ["OracleAS Wireless Edition and XML"](#)
- ["Customizing Presentation with XML and XSQL: Flight Finder"](#)

Business-to-Business (B2B) or Business-to-Consumer (B2C) Messaging

B2B and B2C messaging involves exchanging data between business applications. These applications typically process generated XML documents or a combination of generated and composed XML documents.

See: These B2B Chapters:

- [Chapter 9, "Exchanging XML Data Using Oracle AQ"](#)

When to Use Oracle XML Components: How They Work Together

For descriptions of the Oracle XML components and how they work together see [Chapter 2, "Modeling and Design Issues for Oracle XML Applications"](#) and [Chapter 3, "Oracle XML Developer Kits \(XDKs\) and Components: Overview and General FAQs"](#).

The remaining sections of this manual, describe how to use the Oracle XML components, Oracle development tools, and how to build Web-based, database applications with these tools.

Oracle XML-Enabled Technology Components and Features

Oracle Database is well-suited for building XML database applications. Oracle XML-enabled technology has the following features:

- [Indexing and Searching XML Documents with Oracle Text \(*interMedia Text*\)](#)
- [Messaging Hubs and Middle Tier Components](#)
- [Back-End to Database to Front-End Integration Issues](#)
- [Oracle XDKs Provide the Two Most Common APIs: DOM and SAX](#)
- [The Oracle Suite of Integrated Tools and Components](#)
- [Oracle XML Samples and Demos](#)

Indexing and Searching XML Documents with Oracle Text (*interMedia Text*)

Oracle Text (*interMedia Text*) provides powerful search and retrieval options for XML stored in CLOBs and other documents. It can index and search XML documents and document sections as large as 4 Gigabytes each stored in a column in a table.

Oracle Text XML document searches include hierarchical element containership, doctype discrimination, and searching on XML attributes. These XML document searches can be used in combination with standard SQL query predicates or with other powerful lexical and full-text searching options.

XML documents or document sections saved into text CLOBs in the database can be enabled for indexing by Oracle Text's text-search engine. Developers can pinpoint searches to data within a specific XML hierarchy as well as locate name-value pairs in attributes of XML elements.

Since Oracle Text is seamlessly integrated into the database and the SQL language, developers can easily use SQL to perform queries that involve both structured data and indexed document sections.

See Also:

- [Chapter 8, "Searching XML Data with Oracle Text"](#)
- *Oracle9i Text Reference*

Messaging Hubs and Middle Tier Components

Also included in Oracle XML are the following components:

- **XML-Enabled Messaging Hubs.** These hubs are vital in business-to-business applications that interface with non-Oracle systems. See also [Chapter 9, "Exchanging XML Data Using Oracle AQ"](#).
- **Middle Tier Systems:** XML-enabled application, web, or integrated servers, such as Oracle Integration Server (OIS) and Oracle Application Server.

Oracle JVM (Java Virtual Machine)

Built from the ground up on Oracle Multi-threaded Server (MTS) architecture, Oracle JVM (Jserver) is a Java 1.2 compliant virtual machine that data server shares memory address space. This allows the following:

- Java and XML-processing code to run with in-memory data access speeds using standard JDBC interfaces.
- Natively compile Java byte codes to improve performance of server-side Java, with linear scalability to thousands of concurrent users. Oracle XDK components are pre-loaded and natively compiled.

Oracle JVM supports native CORBA and EJB standards as well as Java Stored Procedures for easy integration of Java with SQL and PL/SQL.

Oracle Application Server

Oracle Application Server (OracleAS), offers services for both intranet and internet Web applications. It is integrated with Oracle and offers advanced services such as data caching and Oracle Portal. OracleAS also provides other services including Oracle Advanced Queueing, Oracle Message Broker, Oracle Workflow, Oracle Reports Services, Dynamic Services, and more.

See Also: <http://otn.oracle.com/products/>

Back-End to Database to Front-End Integration Issues

A key development challenge is integrating back-end ERP and CRM systems from multiple vendors, with systems from partners in their supply chain, and with customized data warehouses.

Such data exchange between different vendors' relational and object-relational databases is simpler using XML.

Oracle XML Technology and Oracle XML-enabled tools, interfaces, and servers provide building blocks for most data and application integration challenges.

Higher Performance Implications

Not only are these building blocks available, but their use results in higher performance implementations for the following reasons:

- Processing database data and XML together on the same server helps eliminate network traffic for data access.
- Exploiting the speed of the Oracle query engine and Oracle JVM, Oracle Application Server, or OIS further enhances data access speed.
- XDK for C components can be used for their native XML capabilities and higher performance

Hence developers can build XML-based Web solutions that integrate Java and database data and facilities in many ways.

Oracle XDKs Provide the Two Most Common APIs: DOM and SAX

Oracle XDKs are implemented in four languages, Java, C, C++, and PL/SQL. The Java version runs directly on Oracle JVM (Java virtual machine). It supports the XML 1.0 specification and is used as a validating or non-validating parser.

The parser provides the two most common APIs that developers need for processing XML documents:

- **DOM 1.0 and 2.0:** W3C-recommended Document Object Model (DOM) interface. This provides a standard way to access and edit a parsed document's element contents.
- **SAX 1.0 and 2.0:** Simple API for XML interface.

For more information, see [Chapter 20, "Using XML Parser for Java"](#). See [Appendix B, "Comparing Oracle XML Parsers and Class Generators by Language"](#), for a comparison of the Oracle XML parsers and generators.

Writing Custom XML Applications

Writing custom applications that process XML documents can be simpler in an Oracle environment. This enables you to write portable standards-based applications and components in your language of choice that can be deployed on any tier.

The XML parser is part of the Oracle Database platform on every operating system where Oracle Database is ported.

Oracle XML Parser is also implemented in PL/SQL. Existing PL/SQL applications can be extended to take advantage of Oracle XML technology.

The Oracle Suite of Integrated Tools and Components

Oracle provides an integrated suite of tools and components for building e-business applications:

- [Oracle JDeveloper and Oracle Business Components for Java \(BC4J\)](#)
- [Oracle Internet File System](#)
- [Oracle Portal](#)
- [Oracle Exchange](#)

This suite of tools ensure that exchanging data and document objects is simplified for application development and that multiple serializations is eliminated.

Oracle JDeveloper and Oracle Business Components for Java (BC4J)

Oracle JDeveloper is an integrated environment for building, deploying, and debugging applications leveraging Java and XML on Oracle. It facilitates working in Java 1.1 or 1.2 with CORBA, EJB, and Java Stored Procedures. With it you can do the following:

- Directly access Oracle XML components to build multitier applications
- Quickly create and debug Java Servlets that serve XML information
- Build portable application logic with JDeveloper and BC4J components

Examples of applications built using Oracle JDeveloper include:

- iProcurement (Self Service Applications) including Self-Service Web-Expensing.
- Online Marketplaces

See [Chapter 11, "Using JDeveloper to Build Oracle XML Applications"](#) for more information on JDeveloper and XML applications.

Oracle Business Components for Java (BC4J) Business Components for Java (BC4J) is an Oracle application framework for encapsulating business logic into reusable libraries of Java components and reusing the business logic through flexible, SQL-based views of information.

Note: Oracle JDeveloper and BC4J are not included with Oracle. Only the BC4J runtime is included. You can download JDeveloper from OTN.

Oracle Internet File System

Access to Oracle Internet File System (*iFS*) facilitates organizing and accessing documents and data using a file- and folder-based model through standard Windows and Internet protocols such as SMB, HTTP, FTP, SMTP, and IMAP4.

iFS facilitates building and administering Web-based applications. It is an application interface for Java and can load a document, such as a Powerpoint file, into Oracle and display the document from a Web server, such as Oracle Application Server or Apache Web Server.

iFS is a simple way for developers to work with XML, where *iFS* serves as the repository for XML. *iFS* automatically parses XML and stores content in tables and columns. *iFS* renders the content when a file is requested delivering select information, for example, on the Web.

For more information see <http://otn.oracle.com/products/ifs/>

Oracle Portal

Oracle Portal can, for example, input XML-based Rich Site Summary (RSS) format documents, and merge the information with an XSL stylesheet. The result can be rendered in a browser. This design efficiently separates the rendition of information from the information itself and allows for easy customization of the look and feel without risk to data integrity.

Oracle Portal is software for building and deploying enterprise portals, the Web sites that power an e-business. The browser interface delivers an organized, personalized view of business information, Web content, and applications needed by each user. It includes site-building and self-service Web publishing functionality

of WebDB 2.2 and adds new enterprise portal features such as single sign-on, personalization, and content classification. Oracle Portal uses Oracle Database and is deployed on and packaged with Oracle Application Server.

Portlets: Portlets are reusable interface components that provide access to Web-based resources. Any Web page, application, business intelligence report, syndicated content feed, hosted software service or other resource can be accessed through a portlet, allowing it to be personalized and managed as a service of Oracle Portal. Companies can create their own portlets and select portlets from third-party portlet providers. Oracle provides a Portal Developer's Kit (PDK) for developers to easily create portlets using PL/SQL, Java, HTML, or XML.

See Also: [Chapter 15, "Using the PDK for Visualizing XML Data in Oracle Portal"](#) for an introduction to Oracle Portal's PDF and URL Services.

Oracle Exchange

The Oracle Exchange platform is based on Oracle Database. It offers all necessary business transactions to support an entire industry's or a company's supply chain. Oracle Exchange is based on Oracle's e-Business Suite, which supports a supply chain from the initial contact with the prospect, to manufacturing planning and execution, to post-sales ongoing service and support.

Oracle Exchange uses XML as its data exchange format and message payload, and Advanced Queuing.

See Also: [Chapter 16, "How Oracle Exchange Uses XML"](#)

XML Gateway

XML Gateway is a set of services that allow you to easily integrate with the Oracle e-Business Suite, to create and consume XML messages triggered by business events. It also integrates with Oracle Advanced Queuing to enqueue/dequeue messages which are then transmitted to/from business partners through any message transport service, including Oracle Message Broker.

See Also: [Chapter 17, "Introducing Oracle XML Gateway"](#)

Metadata API

Metadata API provides a centralized, simple, and flexible means for performing the following tasks:

- Extracting complete definitions of database objects (metadata) as either XML or creation DDL
- Transforming metadata via industry-standard XML Stylesheet Transformation language (XSLT).
- Generating SQL DDL to recreate the database objects

Metadata API is available on Oracle Database whenever the instance is operational. It is not available on Oracle Lite.

See Also: [Chapter 13, "Using Metadata API"](#)

Other XML Initiatives

Besides these tools, the following initiatives are underway.

XML Metadata Interchange (XMI): Managing and Sharing Tools and Data Warehouse Metadata

Support for XML Metadata Interchange (XMI) specification proposed by Oracle, IBM, and Unisys. This enables application development tools and data warehousing tools from Oracle and others to exchange common metadata, ensuring that you can choose any tool without having to modify your application and warehouse design.

Advanced Queueing XML Support: Using the Internet for Reliable, Asynchronous Messaging

Oracle Advanced Queueing (AQ) now allows reliable propagation of asynchronous messages, including messages with XML documents, document sections, or even fragments as their payload, over secure HTTP. This enables dynamic trading and eliminates delays and startup costs to establish inter-company or inter-agency links.

See Also: [Chapter 9, "Exchanging XML Data Using Oracle AQ"](#)

Oracle XML Samples and Demos

This manual contains examples that illustrate the use of Oracle XML components. The examples do not conform to one schema. Where examples are available for download or supplied with the `$ORACLE_HOME/rdbms/demo` or `$ORACLE_HOME/xdk/.../sample`, this is indicated.

What Is Needed to Run Oracle XML Components

Oracle8i and higher includes native support for internet standards, including Java and XML. You can run Oracle XML components and applications built with them inside the database itself using Oracle JServer, a built-in Java Virtual Machine.

Use Oracle Lite to store and retrieve XML data, for devices and applications that require a smaller database footprint.

Oracle XML components can be downloaded for free from
<http://otn.oracle.com/tech/xml>

Requirements for XDK

The following are requirements for XDK for Java and XDK for PL/SQL:

- XDK for Java requires JDK/JRE 1.1 or high VM for Java
- XDK for PL/SQL requires Oracle8x or higher, or PL/SQL cartridge

Requirements are also discussed in the XDK chapters, chapters 19 through 29, and Appendixes C through G.

Which XML Components are Included with Oracle Database and Oracle Application Server?

[Table 1-1](#) lists the XDK component versions included with Oracle Database and Oracle Application Server (OracleAS):

Table 1-1 Oracle Database and OracleAS XDK Component Supplied Versions

XDK Component	Oracle Database Rel. 10g (9.0.4)	OracleAS Rel.--prelliminary version nos. only
XDK for Java	-	-
XML Parser for Java and XSLT Processor	9.0.1.0.0	9.0.1.0.0
XML Schema Processor for Java	9.0.1.0.0	9.0.1.0.0
XML Class Generator for Java	9.0.1.0.0	9.0.1.0.0
XSQL Servlet	9.0.1.0.0	9.0.1.0.0
XML SQL Utility (XSU) for Java	9.0.1.0.0	9.0.1.0.0
XDK for Java Beans	-	-

Table 1–1 Oracle Database and OracleAS XDK Component Supplied Versions(Cont.)

XDK Component	Oracle Database Rel. 10g (9.0.4)	OracleAS Rel.--preliminary version nos. only
XML Transviewer Beans	9.0.1.0.0	9.0.1.0.0
XDK for C	-	-
XML Parser for C and XSLT Processor	9.0.1.0.0	9.0.1.0.0
XML Schema Processor for C	9.0.1.0.0	9.0.1.0.0
XDK for C++	-	-
XML Parser for C++ and XSLT Processor	9.0.1.0.0	9.0.1.0.0
XML Schema Processor for C++	9.0.1.0.0	9.0.1.0.0
XML Class Generator for C++	9.0.1.0.0	9.0.1.0.0
XDK for PL/SQL	-	-
XML Parser for PL/SQL and XSLT Processor	9.0.1.0.0	9.0.1.0.0
XML SQL Utility (XSU) for PL/SQL	9.0.1.0.0	9.0.1.0.0

XML Technical Support

Besides your regular channels of support through your customer representative or consultant, technical support for Oracle XML-enabled technologies is available free through the Discussions option on Oracle Technology Network (OTN):

<http://otn.oracle.com/tech/xml>

You do not need to be a registered user of OTN to post or reply to XML-related questions on the OTN technical discussion forum. To use the OTN technical forum follow these steps:

1. In the left-hand navigation bar, of the OTN site select Support > Discussions.
2. Click on Enter a Technical Forum.
3. Scroll down to the Technologies section. Select XML.
4. Post any questions, comments, requests, or bug reports there.

Download the Latest Software From OTN

You will find the latest information about the Oracle XML components and can download them from OTN:

<http://otn.oracle.com/software/>

At the top, under Download Oracle Products, Drivers, and Utilities, in the Select a Utility or Driver pull down menu, scroll down and select any of the XML utilities listed. For the latest XML Parser for Java and C++, select v2.

Modeling and Design Issues for Oracle XML Applications

This chapter contains the following sections:

- XML Data can be Stored as Generated XML or Composed XML
- Generated XML
- Composed (Authored/Native) XML
- Using a Hybrid XML Storage Approach for Better Mapping Granularity
- Transforming Generated XML
- General XML: Design Issues for Data Exchange Applications
- Sending XML Documents Applications-to-Application
- Loading XML into a Database
- Applications that Use Oracle XML -EnabledTechnology
- Content and Document Management with XML-Enabled Technology
- Business-to-Business and Business-to-Consumer Messaging

XML Data can be Stored as Generated XML or Composed XML

XML data can be stored in Oracle in the following ways:

- **Generated XML**, where the XML data is stored across object-relational tables or as views in the database. This data can then be generated back into XML format, dynamically, when necessary
- **Composed (Authored/Native) XML**, where the XML document is stored as is in CLOBs

Generated XML

XML can be generated from object-relational tables and views. The benefits of using object-relational tables and views as opposed to pure relational structures are discussed below.

Generated XML is used when the XML is an interchange format and existing business data is wrapped in XML structures (tags). This is the most common way of using XML in the database. Here, XML is used only for the interchange process itself and is transient.

Generated XML Examples

Examples of this kind of document include sales orders and invoices, airline flight schedules, and so on.

Oracle, with its object-relational extensions has the ability to capture the structure of the data in the database using object types, object references, and collections. There are two options for storing and preserving the structure of the XML data in an object-relational form:

- Store the attributes of the elements in a relational table and define object views to capture the structure of the XML elements
- Store the structured XML elements in an object table

Once stored generated, in the object-relational form, the data can be easily updated, queried, rearranged, and reformatted as needed using SQL.

Object-Relational Storage for Generated XML Documents

Complex XML documents can be stored as object-relational instances and indexed efficiently. Such instances fully capture and express the nesting and list semantics of XML. With Oracle's extensibility infrastructure, new types of indices, such as path indices, can be created for faster searching through XML documents.

XML SQL Utility (XSU) Stores XML and Converts SQL Query Results into XML

XML SQL Utility (XSU) provides the means to store an XML document by mapping it to the underlying object-relational storage, and conversely, provides the ability to retrieve the object-relational data as an XML document.

XSU converts the result of an SQL query into XML by mapping the query alias or column names into the element tag names and preserving the nesting of object types. The result can be in text or a DOM (Document Object Model) tree. The generation of the latter avoids the overhead of parsing the text and directly realizes the DOM tree.

See Also: [Chapter 7, "XML SQL Utility \(XSU\)"](#)

Composed (Authored/Native) XML

Oracle8i and higher support the storage of large objects or LOBs as character LOBs (CLOB), binary LOBs (BLOB), or externally stored binary files (BFILE). LOBs are used to store composed (Authored/Native) XML documents.

Storing Composed XML Data in CLOBs or BFILES

If the incoming XML documents do not conform to one particular structure, then it might be better to store such documents in CLOBs. For instance, in an XML messaging environment, each XML message in a queue might be of a different structure.

CLOBs store large character data and are useful for storing composed XML documents.

BFILES are external file references and can also be used, although they are more useful for multimedia data that is not accessed often. In this case the XML is stored and managed outside Oracle, but can be used in queries on the server. The metadata for the document can be stored in object-relational tables in the server for fast indexing and access.

Storing an intact XML document in a CLOB or BLOB is a good strategy if the XML document contains static content that will only be updated by replacing the entire document.

- Composed XML examples include written text such as articles, advertisements, books, legal contracts, and so on. Documents of this nature are known as document-centric and are delivered from the database as a whole. Storing this

kind of document intact within Oracle gives you the advantages of an industry-proven database and its reliability over file system storage.

- **Storage Outside the database.** If you choose to store an XML document outside the database, you can still use Oracle features to index, query, and efficiently retrieve the document through the use of BFILES, URLs, and text-based indexing.

Oracle Text Indexing Enables Fine Grain Searching of Element Content

Oracle allows the creation of Oracle Text (*interMedia* Text) indexes on LOB columns, in addition to URLs that point to external documents. This indexing mechanism works for XML data as well.

Oracle8i and Oracle Database recognize XML tags, and section and sub-section text searching within XML elements' content. The result is that queries can be posed on unstructured data and restricted to certain sections or elements within a document.

Oracle Text Example: Searching Text and XML Data Using CONTAINS

This Oracle Text (*interMedia* Text) example presume you have already created the appropriate index.

```
SELECT *
FROM   purchaseXMLTab
WHERE  CONTAINS(po_xml, "street WITHIN addr") >= 1;
```

See Also: [Chapter 8, "Searching XML Data with Oracle Text"](#) for more information on Oracle Text.

Advantages of Using Composed (Authored) XML Storage

CLOB storage is ideal if the structure of the XML document is unknown or dynamic.

Disadvantages of Using Composed XML Storage

Much of the SQL functionality on object-relational columns cannot be exploited. Concurrency of certain operations such as updates may be reduced. However, the exact copy of the document is retained.

Using a Hybrid XML Storage Approach for Better Mapping Granularity

The previous section described the following:

- How structured XML documents (Generated) can be mapped to object-relational instances
- How composed XML documents (Authored) can be stored in LOBs

However, in many cases, you need better control of the mapping granularity.

For example, when mapping a text document, such as a book, in XML, you may not want every single element to be expanded and stored as object-relational. Storing the font and paragraph information for such documents in an object-relational format may not be useful with respect to querying.

On the other hand, storing the whole text document in a CLOB reduces the effective SQL queriability on the entire document.

A Hybrid Approach Allows for User-Defined Storage Granularity

The alternative is to have user-defined granularity for such storage. In the book example, you may want the following:

- To query on top-level elements such as chapter, section, title, and so on. These elements can be stored in object relational tables.
- To query the book's contents in each section. These sections can be stored in a CLOB.

You can specify the granularity of mapping at table definition time. The server can automatically construct the XML from the various sources and generate queries appropriately.

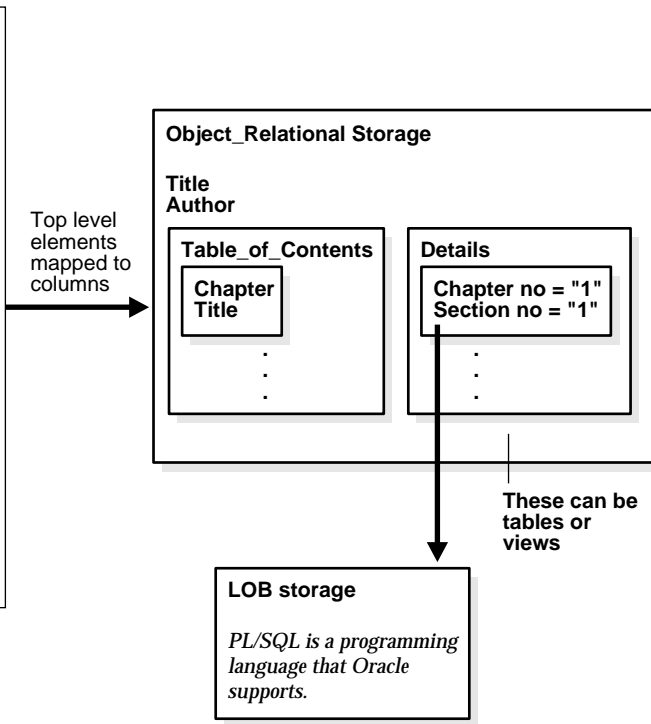
[Figure 2-1](#) illustrates this hybrid approach to XML storage.

Figure 2–1 Hybrid XML Storage Approach: Querying Top Level Elements in Tables While Contents are in a CLOB

XML Document

```

<?xml version = '1.0'?>
<BOOK>
  <TITLE>Oracle PL/SQL</TITLE>
  <AUTHOR>Steve Feuerstein</AUTHOR>
  <TABLE_OF_CONTENTS>
    <CHAPTER>
      <CHAPTER_NUM>1</CHAPTER_NUM>
      <TITLE>Introduction</TITLE>
      <SECTIONS>
        ...
      </SECTIONS>
    </CHAPTER>
    ...
  </TABLE_OF_CONTENTS>
  <DETAILS>
    <CHAPTER no="1">
      <SECTION no="1" name="What is PL/SQL?">
        PL/SQL is a programming language that
        Oracle supports.
      </SECTION>
    </CHAPTER>
  </DETAILS>
</BOOK>
    
```



Hybrid Storage Advantages

The advantages of the hybrid storage approach for storing XML documents are the following:

- It gives the flexibility of storing useful and queryable information in object-relational format while not decomposing the entire document.
- Saves time in reconstructing the document, since the entire document is not broken down.
- Enables text searching on those parts of the document stored in LOBs

Transforming Generated XML

XML generated from the database is in a canonical format that maps columns to elements and object types to nested elements. However, applications might require different representations of the XML document in different circumstances.

When the XML Document Structure Needs Transforming

If an XML document is structured, but the structure of the XML document is not compatible with the structure of the underlying database schema, you must transform the data into the correct format before writing it to the database. You can achieve this in one of the following ways:

- Use XSL stylesheets or other programming approaches
- Store the data-centric XML document as an intact single object
- Define object views corresponding to the various XML document structure and define instead-of triggers to perform the appropriate transformation and update the base data.

Combining XML Documents and Data Using Views

Finally, if you have a combination of structured and unstructured XML data, but still want to view and operate on it as a whole, you can use Oracle views.

Views enable you to construct an object on the fly by combining XML data stored in a variety of ways. You can do the following:

- Store structured data, such as employee data, customer data, and so on, in one location within object-relational tables.
- Store related unstructured data, such as descriptions and comments, within a CLOB.

When you need to retrieve the data as a whole, simply construct the structure from the various pieces of data with the use of type constructors in the view's select statement. XML SQL Utility then enables retrieving the constructed data from the view as a single XML document.

Indexing and Querying Transformations

You may need to create indexes and query on transformed views of an XML document. For example, in an XML messaging environment, there could be

purchase order messages in different formats. You may want to query them canonically, so that a particular query can work across all purchase order messages.

In this case, the query is posed against the transformed view of the documents. You can create functional indexes or use regular views to achieve this.

Indexing Approaches

Native implementation for the `extract()` and `existsNode()` member functions is to parse the XML document, perform path traversal, and extract the fragment. However, this is not a performance-enhancing or scalable solution.

A second approach is to use Oracle Text (*interMedia Text*) indexing.

See Also: [Chapter 8, "Searching XML Data with Oracle Text"](#)

You can also build your own indexing mechanism on an `XMLType` column using the extensibility indexing infrastructure.

See Also: *Oracle9i Data Cartridge Developer's Guide*

XML Schemas and Mapping of Documents

W3C has chartered a schema working group to provide a new, XML based notation for structural schema and datatypes as an evolution of the current Document Type Definition (DTD) based mechanism. XML schemas can be used for the following:

- XML-Schema1: Constraining document structure (elements, attributes, namespaces)
- XMLSchema2: Constraining content (datatypes, entities, notations)

Datatypes themselves can either be primitive (such as bytes, dates, integers, sequences, intervals) or user-defined (including ones that are derived from existing datatypes and which may constrain certain properties -- range, precision, length, mask -- of the basetype.) Application-specific constraints and descriptions are allowed.

XML Schema provides inheritance for element, attribute, and datatype definitions. Mechanisms are provided for URI references to facilitate a standard, unambiguous semantic understanding of constructs. The schema language also provides for embedded documentation or comments.

For example, you can define a simple data type as shown in the following example.

XMLSchema Example 1: Defining a Simple Data Type

This is an example of defining a simple data type in XMLSchema:

```
<datatype name="positiveInteger"
          basetype="integer"/>
  <minExclusive> 0 </minExclusive>
</datatype>
```

It is clear even from the simple example above that XMLSchema provides a number of important new constructs over DTDs, such as a basetype, and a minimum value constraint.

When dynamic data is generated from a database, it is typically expressed in terms of a database type system. In Oracle, this is the object-relational type system described above, which provides for much richness in data types, such as NULL-ness, variable precision, NUMBER(7,2), check constraints, user-defined types, inheritance, references between types, collections of types and so on. XML Schema can capture a wide spectrum of schema constraints that go towards better matching generated documents to the underlying type-system of the data.

XMLSchema Example 2: Map Generated XML Documents to Underlying Schema

Consider the simple Purchase Order type expressed in XML Schema:

```
<type name="Address" >
  <element name="street" type="string" />
  <element name="city" type="string" />
  <element name="state" type="string" />
  <element name="zip" type="string" />
</type>

<type name="Customer">
  <element name="custNo"
          type="positiveInteger"/>
  <element name="custName" type="string" />
  <element name="custAddr" type="Address" />
</type>

<type name="Items">
  <element name="lineItem" minOccurs="0" maxOccurs="*">
    <type>
      <element name="lineItemNo" type="positiveInteger" />
      <element name="lineItemName" type="string" />
      <element name="lineItemPrice" type="number" />
    </type>
  </element>
</type>
```

```
<element name="LineItemQuan">
  <datatype basetype="integer">
    <minExclusive>0</minExclusive>
  </datatype>
</element>
</type>
</element>
</type>

<type name="PurchaseOrderType">
  <element name="purchaseNo"
            type="positiveInteger" />
  <element name="purchaseDate" type="date" />
  <element name="customer" type="Customer" />
  <element name="lineItemList" type="Items" />
</type>
```

These XML Schemas have been deliberately constructed to match closely the Object-Relational purchase order example described above in ["XML Schema Example 2: Map Generated XML Documents to Underlying Schema"](#). The point is to underscore the closeness of match between the proposed constructs of XML Schema with SQL:1999-based type systems. Given such a close match, it is relatively easy to map an XML Schema to a database Object-Relational schema, and map documents that are valid according to the above schema to row objects in the database schema. In fact, the greater expressiveness of XML Schema over DTDs greatly facilitates the mapping.

The applicability of the schema constraints provided by XML Schema is not limited to data-driven applications. There are more and more document-driven applications that exhibit dynamic behavior.

- A simple example might be a memo, which is routed differently based on markup tags.
- A more sophisticated example is a technical service manual for an intercontinental aircraft. Based on complex constraints provided by XML Schema, one can ensure that the author of such a manual always enters a valid part-number, and one might even ensure that part-number validity depends on dynamic considerations such as inventory levels, fluctuating demand and supply metrics, or changing regulatory mandates.

General XML: Design Issues for Data Exchange Applications

This section describes the following XML design issues for applications that exchange data.

- [Generating a Web Form from XML Data Stored in the Database](#)
- [Sending XML Data from a Web Form to the Database](#)

Generating a Web Form from XML Data Stored in the Database

To generate a Web form's infrastructure, you can do the following:

1. Use XML SQL Utility to generate a DTD based on the schema of the underlying table being queried.
2. Use the generated DTD as input to the XML Java Class Generator, which will generate a set of classes based on the DTD elements.
3. Write Java code that use these classes to generate the infrastructure behind a Web-based form. Based on this infrastructure, the Web form can capture user data and create an XML document compatible with the database schema. This data can then be written directly to the corresponding database table or object view without further processing.

Sending XML Data from a Web Form to the Database

One way to ensure that data obtained via a Web form will map to an underlying database schema is to design the Web form and its underlying structure so that it generates XML data based on a schema-compatible DTD. This section describes how to use the XML SQL Utility and the XML Parser for Java to achieve this. This scenario has the following flow:

1. A Java application uses the XML SQL Utility to generate a DTD that matches the expected format of the target object view or table.
2. The application feeds this DTD into the XML Class Generator for Java, which builds classes that can be used to set up the Web form presented to the user.
3. Using the generated classes, the web form is built dynamically by a JavaServer Page, Java servlet, or other component.
4. When a user fills out the form and submits it, the servlet maps the data to the proper XML data structure and the XML SQL Utility writes the data to the database.

You can use the DTD-generation capability of the XML SQL Utility to determine what XML format is expected by a target object view or table. To do this, you can perform a `SELECT * FROM` an object view or table to generate an XML result.

This result contains the DTD information as a separate file or embedded within the `DOCTYPE` tag at the top of the XML file.

Use this DTD as input to the XML Class Generator to generate a set of classes based on the DTD elements. You can then write Java code that use these classes to generate the infrastructure behind a Web-based form. The result is that data submitted via the Web form will be converted to an XML document that can be written to the database.

Sending XML Documents Applications-to-Application

There are numerous ways to transmit XML documents among applications. This section presents some of the more common approaches.

Here you can assume the following:

- The sending application transmits the XML document
- The receiving application receives the XML document

File Transfer. The receiving application requests the XML document from the sending application via FTP, NFS, SMB, or other file transfer protocol. The document is copied to the receiving application's file system. The application reads the file and processes it.

HTTP. The receiving application makes an HTTP request to a servlet. The servlet returns the XML document to the receiving application, which reads and processes it.

Web Form. The sending application renders a Web form. A user fills out the form and submits the information via a Java applet or Javascript running in the browser. The applet or Javascript transmits the user's form in XML format to the receiving application, which reads and processes it. If the receiving application will ultimately write data to the database, the sending application should create the XML in a database compatible format. One way to do this using Oracle XML products is described in the section Sending XML Data from a Web Form to a Database.

Advanced Queuing. An Oracle database sends an XML document via Net Services, HTTP or SMTP, and JDBC to the one or more receiving applications as a message

through Oracle Advanced Queueing (AQ). The receiving applications dequeue the XML message and process it.

See Also:

- [Chapter 9, "Exchanging XML Data Using Oracle AQ"](#)
- *Oracle Integration Server Overview*
- *Oracle9i Application Developer's Guide - Advanced Queuing*

Loading XML into a Database

You can use the following options to load XML data or DTD files into Oracle:

- Use PL/SQL stored procedures for LOB, such as `DBMS_LOB`
- Write Java (Pro*C, C++) custom code
- Use SQL*Loader
- Use Oracle *interMedia*
- XML SQL Utility (XSU)

You can also use Oracle Internet File System (iFS) to put an XML document into the database. However, it does not support DTDs. It does however support XML Schema, the standard that will replace DTDs.

Using SQL*Loader

You can use SQL*Loader to bulk load LOBs.

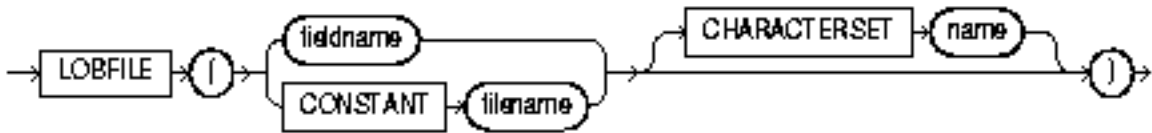
See:

- *Oracle9i Utilities* for a detailed description of using SQL*Loader to load LOBs.
- *Oracle9i Application Developer's Guide - Large Objects (LOBs)*, Chapter 4, "Managing LOBs", "Using SQL*Loader to Load LOBs", for a brief description and examples of using SQL*Loader.

Loading XML Documents Into LOBs With SQL*Loader

Because LOBs can be quite large, SQL*Loader can load LOB data from either the main datafile (inline with the rest of the data) or from LOBFILES. Figure 2–2 shows the LOBFILE syntax.

Figure 2–2 The LOBFILE Syntax



LOB data can be lengthy enough that it makes sense to load it from a LOBFILE. In LOBFILES, LOB data instances are still considered to be in fields (predetermined size, delimited, length-value), but these fields are not organized into records (the concept of a record does not exist within LOBFILES). Therefore, the processing overhead of dealing with records is avoided. This type of organization of data is ideal for LOB loading.

There is no requirement that a LOB from a LOBFILE fit in memory. SQL*Loader reads LOBFILES in 64K chunks. To load physical records larger than 64K, you can use the `READSIZE` parameter to specify a larger size.

It is best to load XMLType columns or columns containing XML data in CLOBs, using LOBFILES.

- **When the XML is valid.** If the XML data in the LOBFILE is large and you know that the data is valid XML, then use direct-path load since it bypasses all the XML validation processing.
- **When the XML needs validating.** If it is imperative that the validity of the XML data be checked, then use conventional path load, keeping in mind that it is not as efficient as a direct-path load.

A conventional path load executes SQL `INSERT` statements to populate tables in an Oracle database. A direct path load eliminates much of the Oracle database overhead by formatting Oracle data blocks and writing the data blocks directly to the database files.

A direct-path load does not compete with other users for database resources, so it can usually load data at near disk speed. Considerations inherent to direct path

loads, such as restrictions, security, and backup implications, are discussed in Chapter 9 of *Oracle9i Utilities*.

Figure 2–3 illustrates SQL*Loader's direct-path load and conventional path loads.

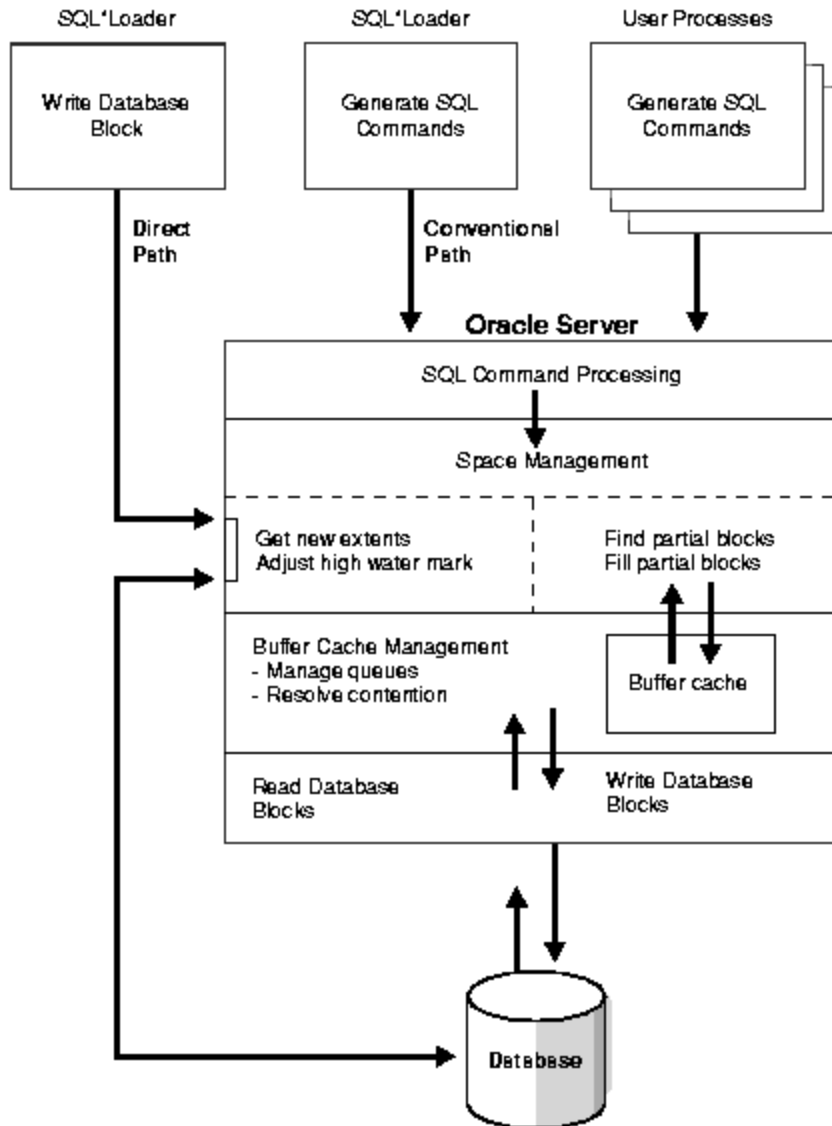
Tables to be loaded must already exist in the database. SQL*Loader never creates tables. It loads existing tables that either already contain data or are empty.

The following privileges are required for a load:

- You must have `INSERT` privileges on the table to be loaded.
- You must have `DELETE` privilege on the table to be loaded, when using the `REPLACE` or `TRUNCATE` option to empty out the table's old data before loading the new data in its place.

See Also: *Oracle9i Utilities*. Chapters 7 and 9 for more information about loading and examples.

Figure 2–3 SQL*Loader: Direct-Path and Conventional Path Loads



Applications that Use Oracle XML -Enabled Technology

There are many potential uses for XML in Internet applications. Two database-centric application areas where Oracle's XML components are well-suited are:

- ["Content and Document Management with XML-Enabled Technology"](#), including customizing data presentation
- ["Business-to-Business and Business-to-Consumer Messaging"](#) for data exchange in inter system or intra system applications

or any combinations of these. This manual focuses on these two application areas, in Part III, ["Data Exchange Using XML"](#) and Part IV, ["Tools and Frameworks for Building Oracle-Based XML Applications"](#), respectively.

Typical scenarios in each of these two application areas are described in this chapter.

Content and Document Management with XML-Enabled Technology

Customizing Presentation of Data

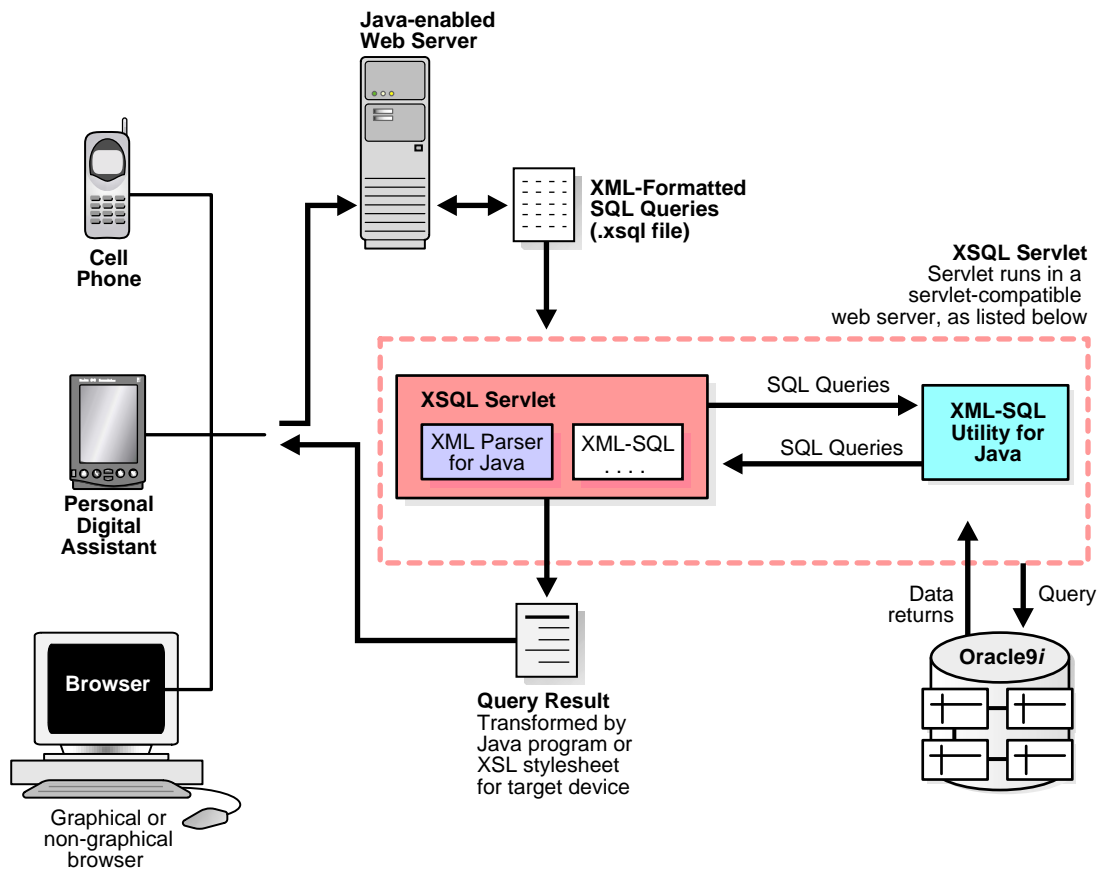
XML is increasingly used to enable customized presentation of data for different browsers, devices, and users. By using XML documents along with XSL stylesheets on either the client, middle-tier, or server, you can transform, organize, and present XML data tailored to individual users for a variety of client devices, including the following:

- Graphical and non-graphical Web browsers
- Personal digital assistants (PDAs), such as the Palm Pilot
- Digital cell phones and pagers

In doing so, you can focus your business applications on business operations, knowing you can accommodate differing output devices easily.

Using XML and XSL also makes it easier to create and manage dynamic Web sites. You can change the look and feel simply by changing the XSL stylesheet, without having to modify the underlying business logic or database code. As you target new users and devices, you can simply design new XSL stylesheets as needed. This is illustrated in [Figure 2-4](#)

Figure 2–4 Content Management: Customizing Your Presentation



See Also: [Chapter 20, "Using XML Parser for Java"](#)

Consider the following content management scenarios that use Oracle's XML components:

- [Scenario 1. Content and Document Management: Publishing Composite Documents Using XML-Enabled OracleTechnology](#)
- [Scenario 2. Content and Document Management: Delivering Personalized Information Using Oracle XML Technology](#)
- [Scenario 3. Content Management: Using Oracle XML Technology to Customize Data Driven Applications](#)

Each scenario includes a brief description of the business problem, solution, main tasks, and Oracle XML components used.

These scenarios are further illustrated in *Oracle9i Case Studies - XML Applications* under the section, "Managing Content and Documents with XML".

Scenario 1. Content and Document Management: Publishing Composite Documents Using XML-Enabled OracleTechnology

Problem

Company X has numerous document repositories of SGML and XML marked up text fragments. Composite documents must be published dynamically.

Solution

The bottom line is that the database application design must begin with a good database design. In other words, Company X must first use good data modeling and design guidelines. Then object views can more readily be created against the data.

Use XMLType to store the documents in XML format, where the relational data is updatable. Use Oracle Internet File System (iFS) as the data repository interface. iFS helps implement XML data repository management and administration tasks.

Company X can use XSL stylesheets to assemble the document sections or fragments and deliver the composite documents electronically to users. One suggested solution is to use Arbortext and EPIC for single sourcing and authoring or multichannel publishing. Multichannel publishing facilitates producing the same document in many different formats, such as HTML, PDF, WORD, ASCII text, SGML, and Framemaker.

See Also: <http://www.arbortext.com> for more information about the Arbortext and EPIC. products.

See [Figure 2-5](#)

Main Tasks Involved

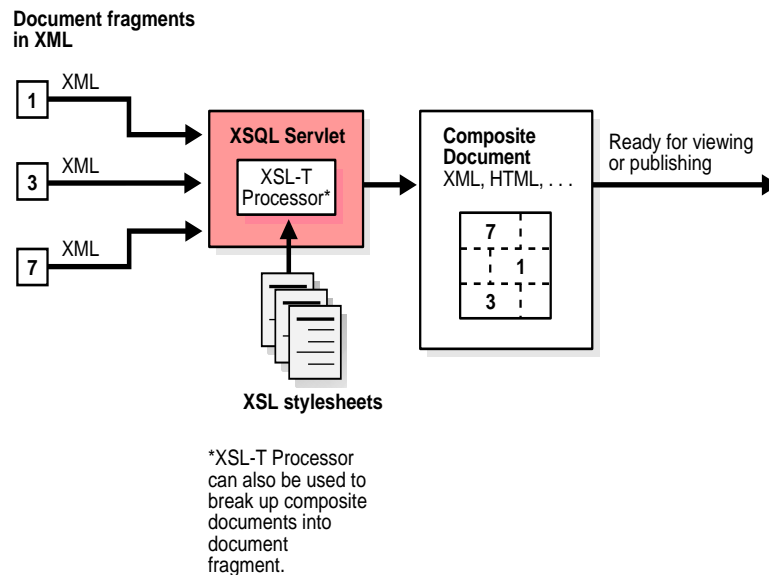
These are the main tasks involved in Scenario 1's solution:

1. Design your database with care. Decide on the XML tags and elements to use.
2. Store these sections or fragments in XMLType columns in CLOBs in the database.
3. Create XSL Stylesheets to render the sections or fragments into complete documents.

Oracle XML Components Used

- XML Parser with XSLT. See [Chapter 20, "Using XML Parser for Java"](#), or [Chapter 24, "Using XML Parser for C"](#).
- XSQL Servlet. See [Chapter 10, "XSQL Pages Publishing Framework"](#).
- XSU to move sections or fragments into and out of the database. See [Chapter 7, "XML SQL Utility \(XSU\)"](#).
- Oracle Text for enhanced data searching applications. See [Chapter 8, "Searching XML Data with Oracle Text"](#).
- For XML storage in XMLType (CLOBs), see [Chapter 5, "Database Support for XML"](#).

Figure 2–5 Scenario 1. Using XSL to Create and Publish Composite Documents



Scenario 2. Content and Document Management: Delivering Personalized Information Using Oracle XML Technology

Problem

A large news distributor receives data from various news sources. This data must be stored in a database and sent to all the distributors and users on demand so that they can view specific and customized news at any time, according to their contract with the news distributor. The distributor uses XSL to normalize and store the data in a database. The stored data is used to back several Websites and portals. These Websites and portals receive HTTP requests from various wired and unwired clients.

Solution

Use XSL stylesheets with the XSQL Servlet to dynamically deliver appropriate rendering to the requesting service. See [Figure 2–6](#)

Main Tasks Involved

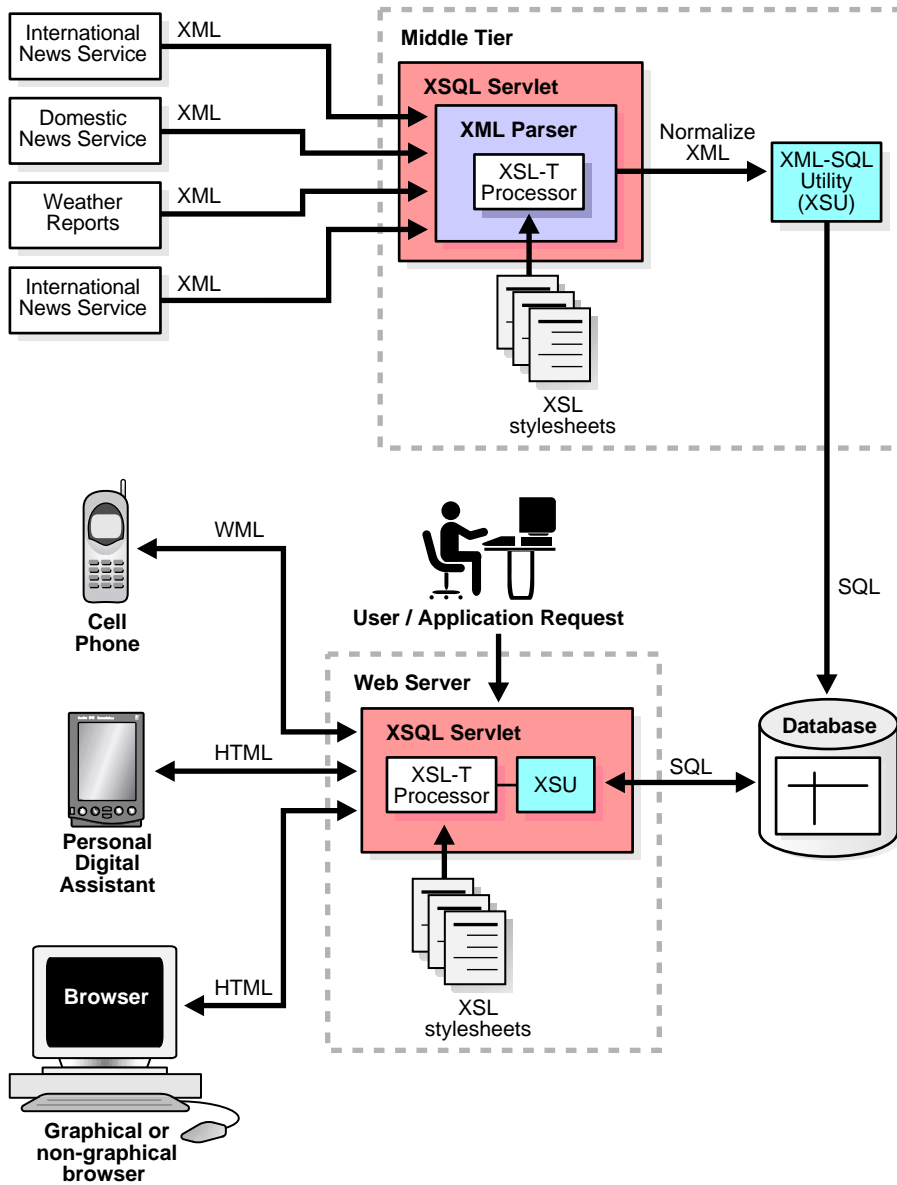
These are the main tasks involved in Scenario 2:

1. Data model for database schema is designed for optimum output.
2. XSL Stylesheets are created for each information source to transform to normalized format. It is then stored in the database.
3. XSL Stylesheets are created along with XSQL pages to present the data on a web site.

Oracle XML Components Used

- XML Parser for Java v2. See [Chapter 20, "Using XML Parser for Java"](#).
- XML SQL Utility (XSU). See [Chapter 7, "XML SQL Utility \(XSU\)"](#).
- XSQL Servlet. See [Chapter 10, "XSQL Pages Publishing Framework"](#).

Figure 2-6 Scenario 2. Oracle XML Components Deliver Customized News Information



Scenario 3. Content Management: Using Oracle XML Technology to Customize Data Driven Applications

Problem

Company X needs data interactively delivered to a thin client.

Solution

Queries are sent from the client to databases whose output is rendered dynamically through one or more XSL stylesheets, for sending to the client application. The data is stored in a relational database in LOBs and materialized in XML.

Oracle XML Components Used

- XML Parser for Java and XSLT Processor. See [Chapter 20, "Using XML Parser for Java"](#) and [Chapter 21, "Using XML Schema Processor for Java"](#).
- For storage of XML in LOBs, see [Chapter 5, "Database Support for XML"](#)

Business-to-Business and Business-to-Consumer Messaging

A challenge for business application developers is to tie together data generated by applications from different vendors and different application domains. Oracle XML-enabled technology makes this kind of data exchange among applications easier to do by focusing on the data and its context without tying it to specific network or communication protocols.

Using XML and XSL transformations, applications can exchange data without having to manage and interpret proprietary or incompatible data formats.

Consider the following business-to-business and business-to-consumer (B2B/B2C) messaging scenarios that use Oracle XML components:

- [Scenario 4. B2B Messaging: Online Multivendor Shopping Cart Design Using XML](#)
- [Scenario 5. B2B Messaging: Using Oracle XML Components and Advanced Queuing for an Online Inventory Application](#)
- [Scenario 6. B2B Messaging: Using Oracle XML-Enabled Technology and AQ for Multi-Application Integration](#)

Each scenario briefly describes the problem, solution, main tasks used to resolve the problem and Oracle XML components used.

Scenario 4. B2B Messaging: Online Multivendor Shopping Cart Design Using XML

Problem

Company X needs to build an online shopping cart, for products coming from various vendors. Company X wants to receive orders online and then based upon which product is ordered, transfer the order to the correct vendor.

Solution

Use XML to deliver an integrated online purchasing application. While a user is completing a new purchase requisition for new hardware, they can go directly to the computer manufacturer's Web site to browse the latest models, configuration options, and prices. The user's site sends a purchase requisition reference number and authentication information to the vendor's Web site.

At the vendor site, the user adds items to their shopping cart, then clicks on a button to indicate that they are done shopping. The vendor sends back the contents of the shopping cart to the Company X's application as an XML file containing the part numbers, quantities, and prices that the user has chosen.

Items from the shopping cart are automatically added to the new purchase requisition as line items.

Customer orders (in XML) are delivered to the appropriate vendor databases for processing. XSL is used to transform and divide the shopping cart for compliant transfers. Data is stored in a relational database and materialized using XML. See [Figure 2-7](#).

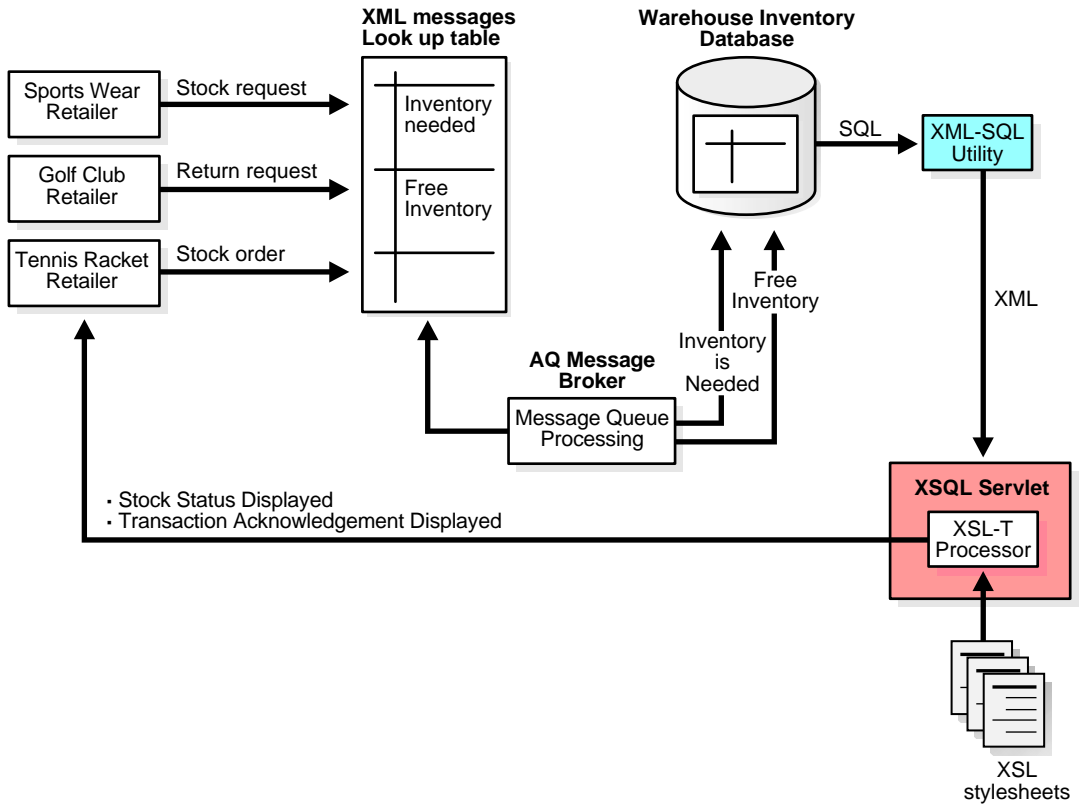
See Also:

- [Chapter 13, "iProcurement Uses XML to Offer Multiple Catalog Products"](#)
 - [Chapter 18, "B2B XML Application: Step by Step"](#)
- for examples of similar implementations

Oracle XML Components Used

- Oracle XML Parser. See [Chapter 20, "Using XML Parser for Java"](#).
- XML SQL Utility. See [Chapter 7, "XML SQL Utility \(XSU\)"](#).
- XSQL Servlet. See [Chapter 10, "XSQL Pages Publishing Framework"](#).

Figure 2-7 Scenario 4. Using Oracle's XML Components for an Online Multivendor Shopping Cart



Scenario 5. B2B Messaging: Using Oracle XML Components and Advanced Queueing for an Online Inventory Application

Problem

A client/server and server/server application stores a data resource and inventory in a database repository. This repository is shared across enterprises. Company X needs to know every time the data resource is accessed, and all the users and customers on the system need to know when and where data is accessed.

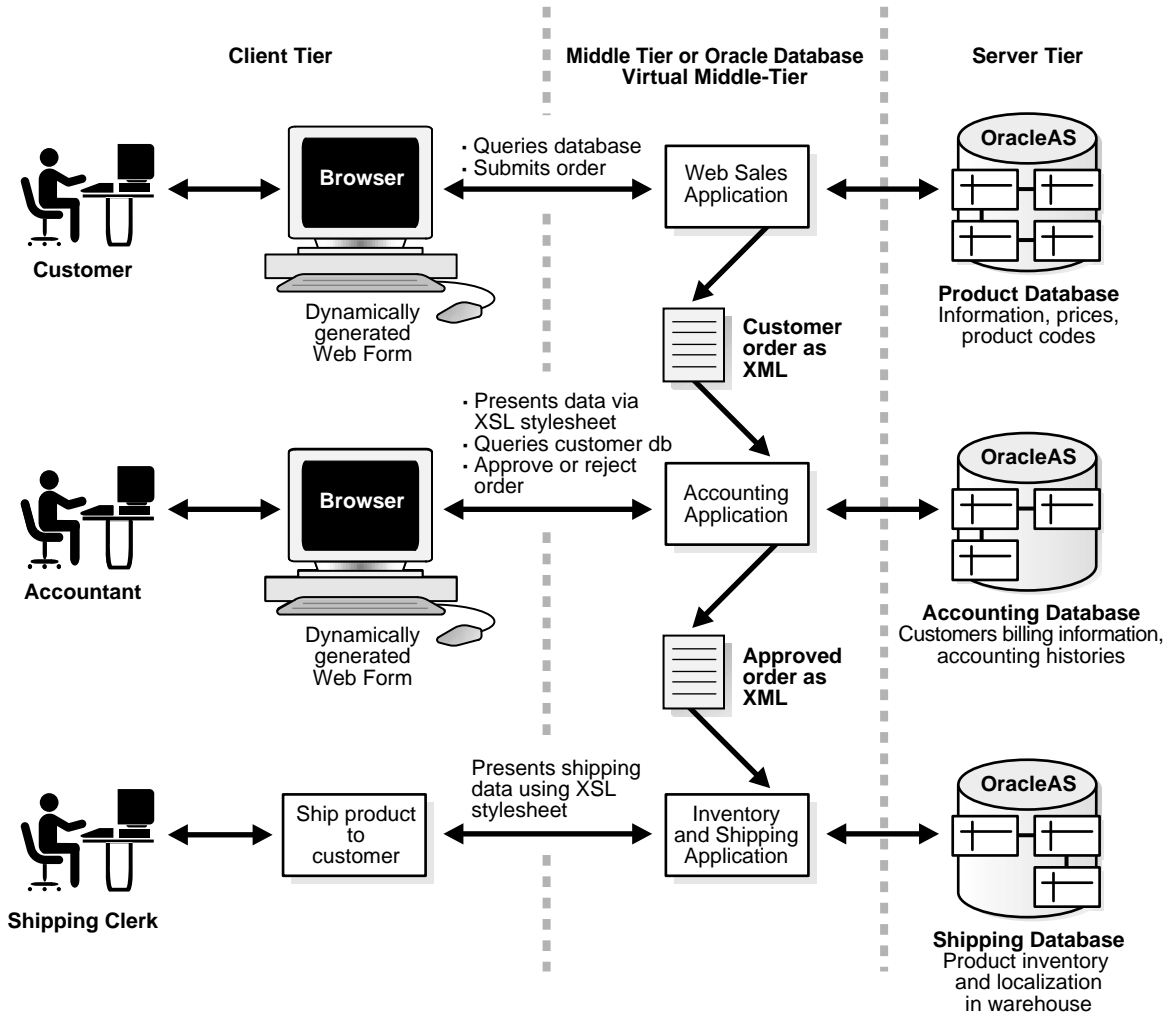
Solution

When a resource is accessed or released this triggers an availability XML message. This in turn transforms the resource, using XSL, into multiple client formats according to need. Conversely, a resource acquisition by one client sends an XML message to other clients, signalling its removal. Messages are stored in LOBs. Data is stored in a relational database and materialized in XML. See [Figure 2-8](#).

Oracle XML Components Used

- XML Parser and XSLT Processor. See [Chapter 20, "Using XML Parser for Java"](#), [Chapter 24, "Using XML Parser for C"](#), [Chapter 26, "Using XML Parser for C++"](#), or [Chapter 29, "Using XML Parser for PL/SQL"](#).
- For storage of XML data in LOBs. See [Chapter 5, "Database Support for XML"](#).

Figure 2–8 Scenario 5. Using Oracle’s XML Components and Advanced Queueing in an Online Inventory Application



Scenario 6. B2B Messaging: Using Oracle XML-Enabled Technology and AQ for Multi-Application Integration

Problem

Company X needs several applications to communicate and share data to integrate the business work flow and processes.

Solution

XML is used as the message payload. It is transformed via the XSLT Processor, enveloped and routed accordingly. The XML messages are stored in an AQ Broker Database in LOBs. Oracle Workflow is used to facilitate management of message and data routing and transformation. This solution also utilizes content management, here presentation customization using XSL stylesheets. See [Figure 2-9](#).

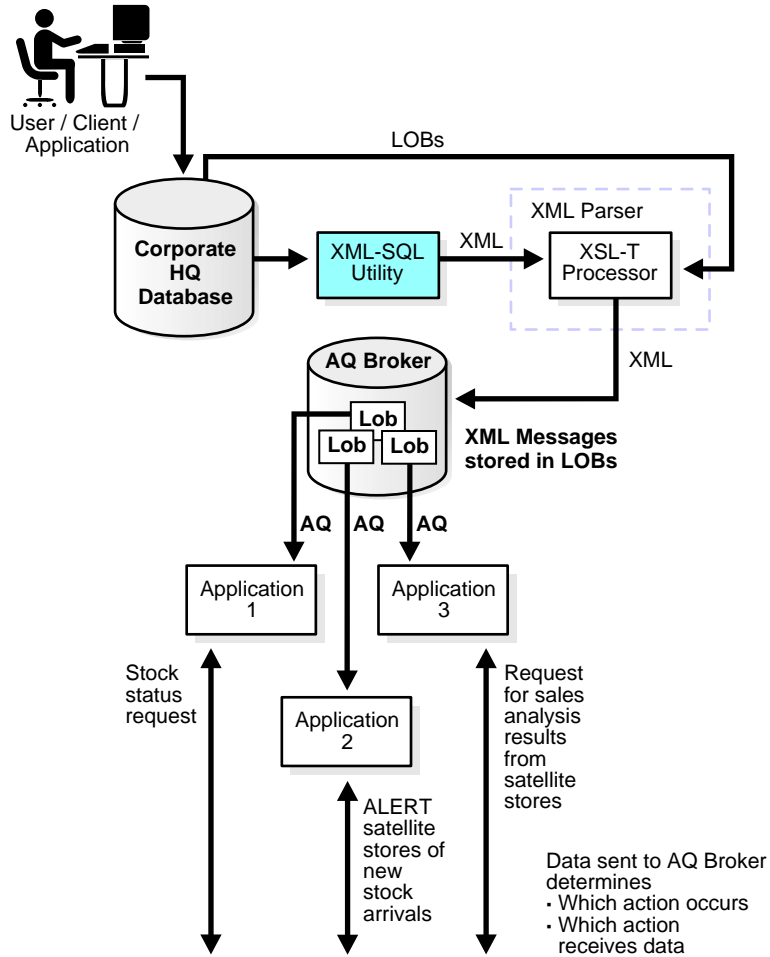
Main Tasks Involved

1. The user or application places a request. The resulting data is pulled from the corporate database using XSU.
2. Data is transformed by XSLT Processor and sent to the AQ Broker.
3. AQ Broker reads this message and determines accordingly what action is needed. It issues the appropriate response to Application 1, 2, and 3, for further processing.

Oracle XML Components Used

- XML Parser and XSLT Processor. See [Chapter 20, "Using XML Parser for Java"](#).
- XML SQL Utility (XSU). See [Chapter 7, "XML SQL Utility \(XSU\)"](#).
- Advanced Queueing. See [Chapter 9, "Exchanging XML Data Using Oracle AQ"](#).

Figure 2–9 Scenario 6. Using Oracle’s XML Components and Advanced Queueing in for Multi-Application Integration



Oracle XML Developer Kits (XDKs) and Components: Overview and General FAQs

This chapter contains the following sections:

- [Oracle XML Components: Overview](#)
- [Development Tools and Other XML-Enabled Features](#)
- [XML Parsers](#)
- [XSL Transformation \(XSLT\) Processor](#)
- [XML Class Generator](#)
- [XML Transviewer Java Beans](#)
- [Oracle XSQL Page Processor and Servlet](#)
- [Oracle XML SQL Utility \(XSU\)](#)
- [Oracle Text](#)
- [Oracle XML Components: Generating XML Documents](#)
- [Using Oracle XML Components to Generate XML Documents: Java](#)
- [Using Oracle XML Components to Generate XML Documents: C](#)
- [Using Oracle XML Components to Generate XML Documents: C++](#)
- [Using Oracle XML Components to Generate XML Documents: PL/SQL](#)
- [Frequently Asked Questions \(FAQs\): Oracle XML-Enabled Technology](#)

Oracle XML Components: Overview

This chapter provides an overview of Oracle's XML components.

Oracle provides several components, utilities, and interfaces you can use to take advantage of XML technology in building your Web-based database applications. Which components you use depends on your application requirements, programming preferences, development, and deployment environments.

The following XML components are provided with Oracle and Oracle Application Server:

- **XML Developer's Kits (XDKs).** There are Oracle XDKs for Java, C, C++, and PL/SQL. These development kits contain building blocks for reading, manipulating, transforming, and viewing XML documents. Oracle XDKs are fully supported and come with a commercial redistribution license. [Table 3-1](#) lists the XDK components.
- **XML SQL Utility (XSU).** This utility, for Java and PL/SQL: Generates and stores XML data to and from the database from SQL queries or result sets or tables. It achieves data transformation, by mapping canonically any SQL query result to XML and vice versa.

The following figures schematically illustrate how the XDK components can be used to generate XML:

- [Figure 3-7, "Generating XML Documents Using XDK for Java"](#)
- [Figure 3-8, "Generating XML Documents Using XDK for C"](#)
- [Figure 3-9, "Generating XML Documents Using XDK for C++"](#)
- [Figure 3-10, "Generating XML Documents Using XDK for PL/SQL"](#)

Table 3-1 XDK Component Descriptions

XDK Component	Languages	Description
XML Parser	Java, C, C++, PL/SQL	Creates and parses XML using industry standard DOM and SAX interfaces.
XSLT Processor	Java, C, C++, PL/SQL	Transforms or renders XML into other text-based formats such as HTML and WML
XML Schema Processor	Java, C, C++	Allows the use of XML simple and complex datatypes by means of your XML Schema definitions.
XML Class Generator	Java, C++	Automatically generates Java and C++ classes from DTDs and XML Schemas to send XML data from Web forms or applications.

Table 3–1 XDK Component Descriptions

XDK Component	Languages	Description
XML Transviewer Java Bean	Java	View and transform XML documents and data via Java components.
XML SQL Utility (XSU)	Java, PL/SQL	Generates XML documents, DTDs, and XML Schemas from SQL queries.
XSQL Servlet	Java	Combines XML, SQL, and XSLT in the server to deliver dynamic Web content.

Development Tools and Other XML-Enabled Features

The following list includes Oracle's XML-enabled development tools:

- **Oracle Text (*interMedia Text/Context*):** A querying, search and retrieval tool, described in [Chapter 8, "Searching XML Data with Oracle Text"](#).
- **Oracle JDeveloper9i and BC4J:** JDeveloper9i is an integrated development tool for building Java web-based applications. Oracle Business Components for Java (BC4J) is a Java, XML-powered framework that enables productive development, portable deployment, and flexible customization of multitier, database-savvy applications from reusable business components. These applications can be deployed as CORBA Server Objects or EJB Session Beans on enterprise-scale server platforms supporting Java technology.

See Also:

- [Chapter 11, "Using JDeveloper to Build Oracle XML Applications"](#)
- [Chapter 12, "Building BC4J and XML Applications"](#)
- **Oracle Internet File System (iFS):** An application interface in which data can be viewed as documents and the documents can be treated as data. *iFS* is a simple way for developers to work with XML, where *iFS* serves as the repository for XML. *iFS* can perform the following tasks on XML documents:
 - Automatically parse XML and store content in tables and columns
 - Render the XML file's content

See Also: *Oracle9i Case Studies - XML Applications*, the chapter, "Using Internet File System (iFS) to Build XML Applications".

- **Oracle Reports.** Oracle Reports Developer and Reports Server enable you to build and publish high-quality, dynamically generated Web reports. Each major task is expedited by the use of a wizard, while the use of report templates and a live data preview allows for easy customization of the report structure. Reports can be published throughout the enterprise via a standard Web browser, in any chosen format, including HTML, HTML Cascading Style Sheets (HTML CSS), Adobe's Portable Document Format (PDF), delimited text, Rich Text Format (RTF), PostScript, PCL, or XML. Reports can be integrated with Oracle Portal (webDB).
- You can schedule reports to run periodically and update the information in an Oracle Portal site. Reports can also be personalized for a user.
- Oracle Reports Developer is part of Oracle's e-business intelligence solution, and integrates with Oracle Discoverer and Oracle Express.

See Also: [Chapter 14, "OracleAS Reports Services and XML"](#)

XDK for Java

XDK for Java is composed of the following components:

- **XML Parser for Java.** Creates and parses XML using industry standard DOM and SAX interfaces. Includes an **XSL Transformation (XSLT) Processor** that transforms XML to XML or other text-based formats, such as HTML.
- **XML Schema Processor for Java.** Supports simple and complex types and is built on the Oracle XML Parser for Java v2.
- **XML Class Generator for Java.** Creates source files from an XML DTD or XML Schema definition.
- **XSQL Servlet.** Processes SQL queries embedded in an XSQL file, xxxx.xsql. Returns results in XML format. Uses XML SQL Utility and XML Parser for Java.
- **XML SQL Utility (XSU) for Java.** Enables you to transform data retrieved from object-relational database tables or views into XML, extract data from an XML document and:
 - Use canonical mapping to insert data into appropriate columns or attributes of a table or a view
 - Apply this data to update or delete values of the appropriate columns or attributes

XDK for Java Beans

XDK for Java Beans is composed of the following component:

- **XML Transviewer Java Beans.** View and transform XML documents and data through Java

XDK for C

XDK for C is composed of the following component:

- **XML Parser for C:** Creates and parses XML using industry standard DOM and SAX interfaces. Includes an **XSL Transformation (XSLT) Processor** that transforms XML to XML or other text-based formats, such as HTML.

XDK for C++

XDK for C++ is composed of the following:

- **XML Parser for C++.** Creates and parses XML using industry standard DOM and SAX interfaces. Includes an **XSL Transformation (XSLT) Processor** that transforms XML to XML or other text-based formats, such as HTML.
- **XML Schema Processor for C++.** A companion component to XML Parser for C++. It allows support for simple and complex datatypes in XML applications with **Oracle Database**. The Schema Processor supports the XML Schema Working Draft.
- **XML C++ Class Generator:** Creates source files from an XML DTD or XML Schema definition.

XDK for PL/SQL

XDK for PL/SQL is composed of the following:

- **XML Parser for PL/SQL:** Creates and parses XML using industry standard DOM and SAX interfaces. Includes an **XSL Transformation (XSLT) Processor** that transforms XML to XML or other text-based formats, such as HTML.
- **XML SQL Utility (XSU) for PL/SQL.** Enables you to transform data retrieved from object-relational database tables or views into XML, extract data from an XML document and:
 - Use canonical mapping to insert data into appropriate columns or attributes of a table or a view

- Apply this data to update or delete values of the appropriate columns or attributes

XML Parsers

The Oracle XML parser includes implementations in C, C++, PL/SQL, and Java for the full range of platforms on which Oracle Database runs.

Based on conformance tests, `xml.com` ranked the Oracle parser in the top two validating parsers for its conformance to the XML 1.0 specification, including support for both SAX and DOM interfaces. The SAX and DOM interfaces conform to the W3C recommendations 2.0.

Version 2 (v2) of the Oracle XML parser provides integrated support for the following features:

- XPath. XPath is the W3C recommendation that specifies the data model and grammar for navigating an XML document utilized by XSLT, XLink and XML Query
- Incremental XSL transformation of document nodes. XSL transformations are compliant with version 1.0 of the W3C recommendations. This support enables the following:
 - Transformations of XML documents to another XML structure
 - Transformations of XML documents to other text-based formats

The parsers are available on all Oracle platforms.

[Figure 3-1](#) illustrates the Oracle XML Parser for Java. [Figure 3-2](#) illustrates the Oracle XML parsers' overall functionality.

See Also: [Chapter 20, "Using XML Parser for Java"](#) and [Appendix C, "XDK for Java: Specifications and Cheat Sheets"](#).

Figure 3–1 Oracle XML Parser for Java

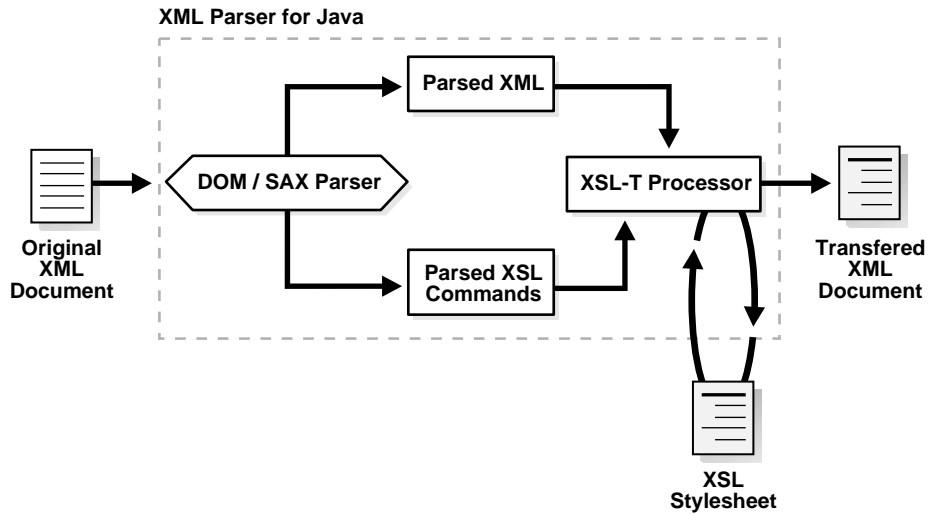
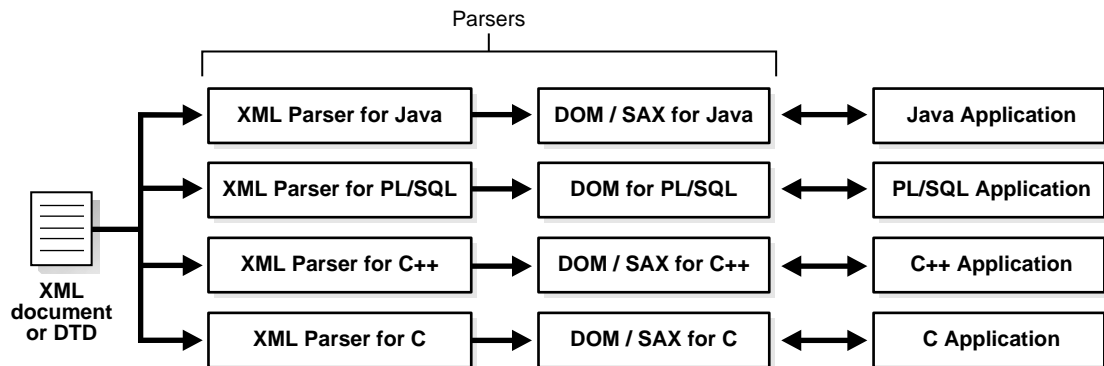


Figure 3–2 The XML Parsers: Java, C, C++, PL/SQL



XSL Transformation (XSLT) Processor

The Oracle XSLT engine fully supports the W3C 1.0 XSL Transformations recommendation. It has the following features:

- Enables standards-based transformation of XML information inside and outside the database on any platform.
- Supports Java extensibility and for additional performance comes natively compiled from Oracle8i Release 3 (8.1.7) and higher.

The Oracle XML Parsers, Version 2 include an integrated XSL Transformation (XSLT) Processor for transforming XML data using XSL stylesheets. Using the XSLT processor, you can transform XML documents from XML to XML, HTML, or virtually any other text-based format.

How to use the XSLT Processor is described in [Chapter 20, "Using XML Parser for Java"](#).

See Also: [Appendix C, "XDK for Java: Specifications and Cheat Sheets"](#).

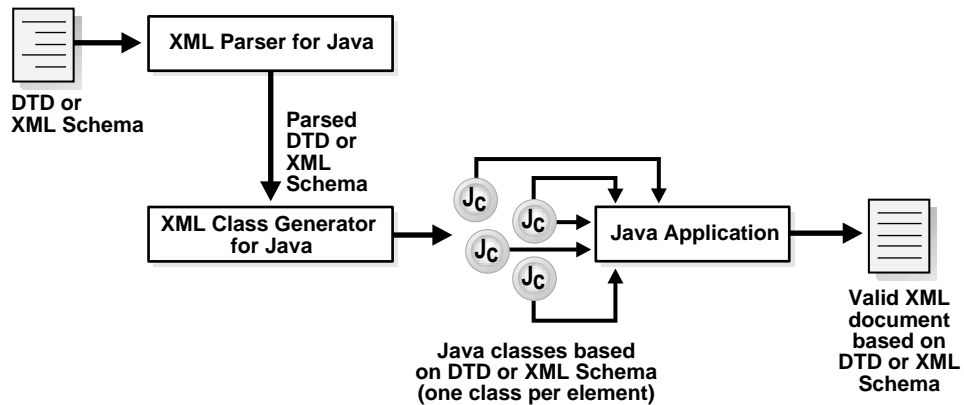
XML Class Generator

XML Class Generator creates a set of Java or C++ classes for creation of XML documents corresponding to an input DTD or XML Schema. [Figure 3-3](#) shows Oracle XML Class Generator functionality.

How to use the XML Class Generators is described in the following chapters:

- [Chapter 22, "XML Class Generator for Java"](#)
- [Chapter 28, "Using XML C++ Class Generator"](#)

Figure 3–3 Oracle XML Java Class Generator



XML Transviewer Java Beans

Oracle XML Transviewer Java Beans are a set of XML components that constitute XML for Java Beans. These are used for Java applications or applets to view and transform XML documents.

They are visual and non-visual Java components that are integrated into Oracle JDeveloper to enable the fast creation and deployment of XML-based database applications. In this release, the following four beans are available:

- **DOM Builder Bean.** This wraps the Java XML (DOM) parser with a bean interface, allowing multiple files to be parsed at once (asynchronous parsing). By registering a listener, Java applications can parse large or successive documents having control return immediately to the caller.
- **XML Source Viewer Bean.** This bean extends JPanel by enabling the viewing of XML documents. It improves the viewing of XML and XSL files by color-highlighting XML and XSL syntax. This is useful when modifying an XML document with an editing application. Easily integrated with the DOM Builder Bean, it allows for pre-parsing and post-parsing and validation against a specified DTD.
- **XML Tree Viewer Bean.** This bean extends JPanel by enabling viewing XML documents in tree form with the ability to expand and collapse XML parsers. It

displays a visual DOM view of an XML document, enabling users to easily manipulate the tree with a mouse to hide or view selected branches.

- **XSL Transformer Bean.** This wraps the XSLT Processor with a bean interface and performs XSL transformations on an XML document based on an XSL stylesheet. It enables users to transform an XML document to almost any text-based format including XML, HTML and DDL, by applying an XSL stylesheet. When integrated with other beans, this bean enables an application or user to view the results of transformations immediately. This bean can also be used as the basis of a server-side application or servlet to render an XML document, such as an XML representation of a query result, into HTML for display in a browser.
- **XML TransPanel Bean.** This bean uses the other beans to create a sample application which can process XML files. This bean includes a file interface to load XML documents and XSL stylesheets. It uses the beans as follows:
 - Visual beans to view and edit files
 - Transformer bean to apply the stylesheet to the XML document and view the output
- DBAccess Bean.
- DBViewer Bean.

As standard Java Beans, they can be used in any graphical Java development environment, such as Oracle JDeveloper. The Oracle XML Transviewer Beans functionality is described in [Chapter 23, "Using XML Transviewer Beans"](#).

Oracle XSQL Page Processor and Servlet

XSQL Servlet is a tool that processes SQL queries and outputs the result set as XML. This processor is implemented as a Java servlet and takes as its input an XML file containing embedded SQL queries. It uses XML Parser for Java, XML- SQL Utility, and Oracle XSL Transformation (XSLT) Engine to perform many of its operations.

You can use XSQL Servlet to perform the following tasks:

- Build dynamic XML datapages from the results of one or more SQL queries and serve the results over the Web as XML datagrams or HTML pages using server-side XSLT transformations.
- Receive XML posted to your web server and insert it into your database.

Servlet Engines that Support XSQL Servlet

XSQL Servlet has been tested with the following servlet engines:

- Allaire JRun 2.3.3
- Apache 1.3.9 with JServ 1.0 and 1.1
- Apache 1.3.9 with Tomcat 3.1 Beta1 Servlet Engine
- Apache Tomcat 3.1 Beta1 Web Server + Servlet Engine
- Caucho Resin 1.1
- NewAtlanta ServletExec 2.2 for IIS/PWS 4.0
- Oracle9i Lite Web-to-Go Server
- Oracle Application Server 4.0.8.1 (with JSP Patch)
- Oracle8i 8.1.7 Beta Aurora and Oracle9i Servlet Engine and higher
- Sun JavaServer Web Development Kit (JSWDK) 1.0.1 Web Server

JavaServer Pages Platforms that Support XSQL Servlet

JavaServer Pages can use `<jsp:forward>` or `<jsp:include>` to collaborate with XSQL Pages as part of an application. The following JSP platforms have been tested to support XSQL Servlet:

- Apache 1.3.9 with Tomcat 3.1 Beta1 Servlet Engine
- Apache Tomcat 3.1 Beta1 Web Server + Tomcat 3.1 Beta1 Servlet Engine
- Caucho Resin 1.1 (Built-in JSP 1.0 Support)
- NewAtlanta ServletExec 2.2 for IIS/PWS 4.0 (Built-in JSP 1.0 Support)
- Oracle9i Lite Web-to-Go Server with Oracle JSP 1.0
- Oracle8i 8.1.7 Beta Aurora and Oracle9i Servlet Engine with Oracle JSP 1.0 and higher
- Any Servlet Engine with Servlet API 2.1+ and Oracle JSP 1.0

In general, it should work with the following:

- Any servlet engine supporting the Servlet 2.1 specification or higher
- Oracle JSP 1.0 reference implementation or functional equivalent from another vendor

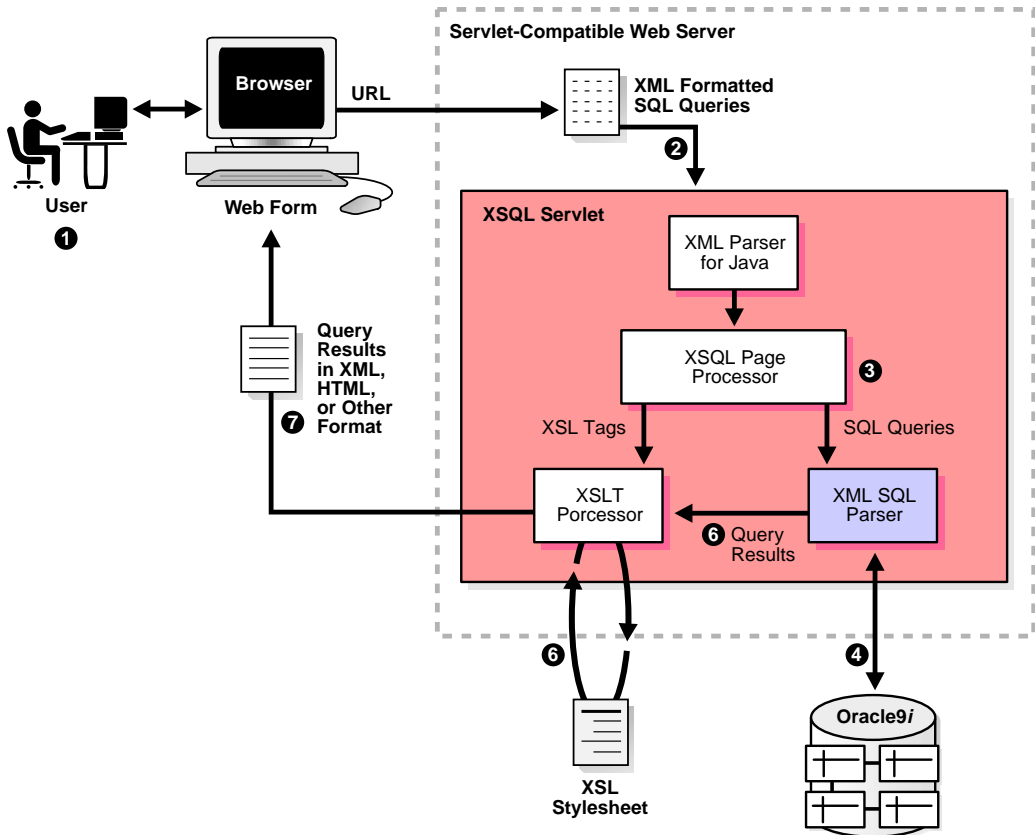
XSQL Servlet is a tool that processes SQL queries and outputs the result set as XML. This processor is implemented as a Java servlet and takes as its input an XML file containing embedded SQL queries. It uses XML Parser for Java and XML SQL Utility to perform many of its operations.

[Figure 3–4](#) shows how data flows from a client, to the servlet, and back to the client. The sequence of events is as follows:

1. The user enters a URL through a browser, which is interpreted and passed to the XSQL Servlet through a Java Web Server. The URL contains the name of the target XSQL file (.xsql) and optionally, parameters, such as values and an XSL stylesheet name. Alternatively, the user can invoke the XSQL Servlet from the command line, bypassing the browser and Java web server.
2. The servlet passes the XSQL file to the XML Parser for Java, which parses the XML and creates an API for accessing the XML contents.
3. The page processor component of the servlet uses the API to pass XML parameters and SQL statements (found between `<query></query>` tags) to XML SQL Utility. The page processor also passes any XSL processing statements to the XSLT Processor.
4. XML SQL Utility sends the SQL queries to the underlying Oracle Database, which returns the query results to the utility.
5. XML SQL Utility returns the query results to the XSLT Processor as XML formatted text. Results are embedded in the XML file in the same location as the original `<query>` tags.
6. If desired, the query results and any other XML data are transformed by the XSLT processor using a specified XSL stylesheet. The data can be transformed to HTML or any other format defined by the stylesheet. The XSLT processor can selectively apply different stylesheets based on the type of client that made the original URL request. This `HTTP_USER_AGENT` information is obtained from the client through an HTTP request.
7. The XSLT Processor passes the completed document back to the client browser for presentation to the user.

See Also: [Chapter 10, "XSQL Pages Publishing Framework"](#)

Figure 3-4 Oracle XSQL Page Processor and Servlet Functional Diagram



Oracle XML SQL Utility (XSU)

Oracle XML SQL Utility (XSU) supports Java and PL/SQL.

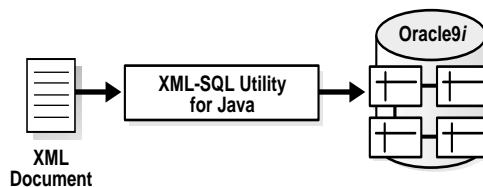
- **XML SQL Utility** is comprised of core Java class libraries for automatically and dynamically rendering the results of arbitrary SQL queries into canonical XML. It includes the following features:
 - Supports queries over richly-structured user-defined object types and object views.
 - Supports automatic XML Insert of canonically-structured XML into any existing table, view, object table, or object view. By combining with XSLT transformations, virtually any XML document can be automatically inserted into the database.

XML SQL Utility Java classes can be used for the following tasks:

- Generate from an SQL query or Result set object a text or XML document, a Document Object Model (DOM), Document Type Definition (DTD), or XML Schema.
- Load data from an XML document into an existing database schema or view.
- **XML SQL Utility for PL/SQL** is comprised of a PL/SQL package that wraps the XML SQL Utility for Java.

Figure 3–5 shows the Oracle XML SQL Utility overall functionality.

Figure 3–5 Oracle XML SQL Utility Functional Diagram



XML SQL Utility for Java consists of a set of Java classes that perform the following tasks:

- Pass a query to the database and generate an XML document (text or DOM) from the results or the DTD which can be used for validation.

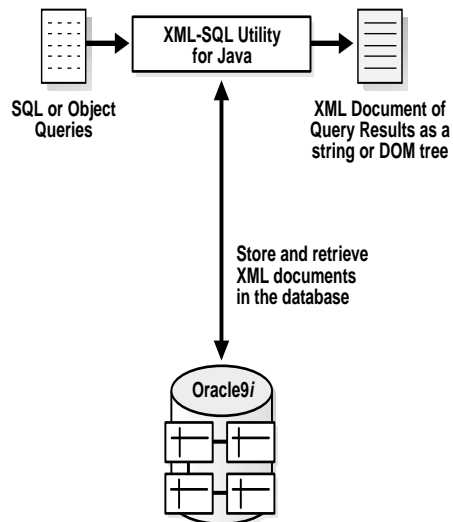
- Write XML data to a database table

See Also: [Chapter 7, "XML SQL Utility \(XSU\)"](#)

Generating XML from Query Results

[Figure 3-6](#) shows how XML SQL Utility processes SQL queries and returns the results as an XML document.

Figure 3-6 *XML-SQL Utility Processes SQL Queries and Returns the Result as an XML Document*



XML Document Structure: Columns Are Mapped to Elements

The structure of the resulting XML document is based on the internal structure of the database schema that returns the query results:

- Columns are mapped to top level elements
- Scalar values are mapped to elements with text-only content
- Object types are mapped to elements with attributes appearing as sub-elements
- Collections are mapped to lists of elements

XSU Generates the XML Document as a String or DOM Element Tree

The XML SQL Utility (XSU) generates either of the following:

- A string representation of the XML document. Use this representation if you are returning the XML document to a requester.
- An in-memory XML DOM tree of elements. Use this representation if you are operating on the XML programmatically, for example, transforming it using the XSLT Processor using DOM methods to search or modify the XML in some way.

XSU Generates a DTD Based on Queried Table's Schema

You can also use the XML SQL Utility (XSU) to generate a DTD based on the schema of the underlying table or view being queried. You can use the generated DTD as input to the XML Class Generator for Java or C++. This generates a set of classes based on the DTD elements. You can then write code that uses these classes to generate the infrastructure behind a Web-based form. See also "[XML Class Generator](#)".

Based on this infrastructure, the Web form can capture user data and create an XML document compatible with the database schema. This data can then be written directly to the corresponding database table or object view without further processing.

See Also: [Chapter 7, "XML SQL Utility \(XSU\)"](#) and *Oracle9i Case Studies - XML Applications*, the chapter, "[B2B XML Application: Step by Step](#)", for more information about this approach.

Note: To write an XML document to a database table, where the XML data does not match the underlying table structure, transform the XML document before writing it to the database. For techniques on doing this, see [Chapter 7, "XML SQL Utility \(XSU\)"](#).

Oracle Text

Oracle Text (*interMedia Text*) extends Oracle Database by indexing any text or documents stored in Oracle. Use Oracle Text to perform searches on XML documents stored in Oracle by indexing the XML as plain text, or as document

sections for more precise searches, such as `find Oracle WITHIN title where title` is a section of the document.

Developer

See Also: [Chapter 8, "Searching XML Data with Oracle Text"](#), for more information on using Oracle Text and XML.

Oracle XML Components: Generating XML Documents

[Figure 3-7](#) through [Figure 3-10](#) illustrate the relationship of the Oracle XML components and how they work together to generate XML documents from Oracle via a SQL query. The options are depicted according to language used:

- Java
- C
- C++
- PL/SQL

Using Oracle XML Components to Generate XML Documents: Java

[Figure 3-7](#) shows the Oracle XML Java components and how they can be used to generate an XML document. Available XML Java components are:

- XDK for Java:
 - XML Parser for Java, Version 2 including the XSLT
 - XML Schema Processor for Java
 - XML Class Generator for Java
 - XSQL Servlet
 - XML Transviewer Beans
- XML SQL Utility (XSU) for Java

In the Java environment, when a user or client or application sends a query (SQL), there are three possible ways of processing the query using the Oracle XML components:

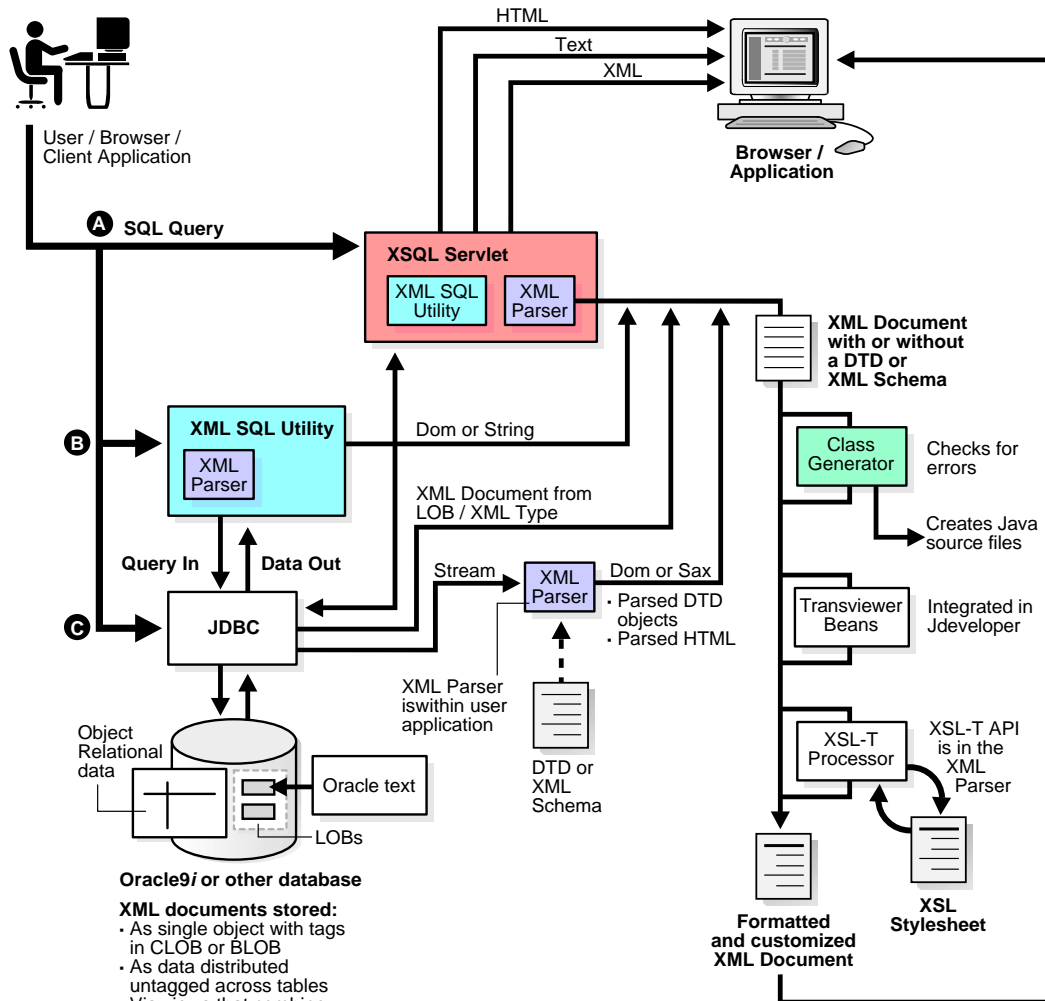
- By the XSL Servlet (this includes using XSU and XML Parser)
- Directly by the XSU (this includes XML Parser)

- Directly by JDBC which then accesses XML Parser

Regardless of which way the stored XML data is generated from the database, the resulting XML document output from the XML Parser is further processed, depending on what you or your application needs it for.

The XML document is formatted and customized by applying stylesheets and processed by the XSLT.

Figure 3-7 Generating XML Documents Using XDK for Java



Using Oracle XML Components to Generate XML Documents: C

Figure 3–8 shows the Oracle XML C language components used to generate an XML document. The XML components are:

- XML Parser/XSLT Processor for C
- XML Schema Processor for C

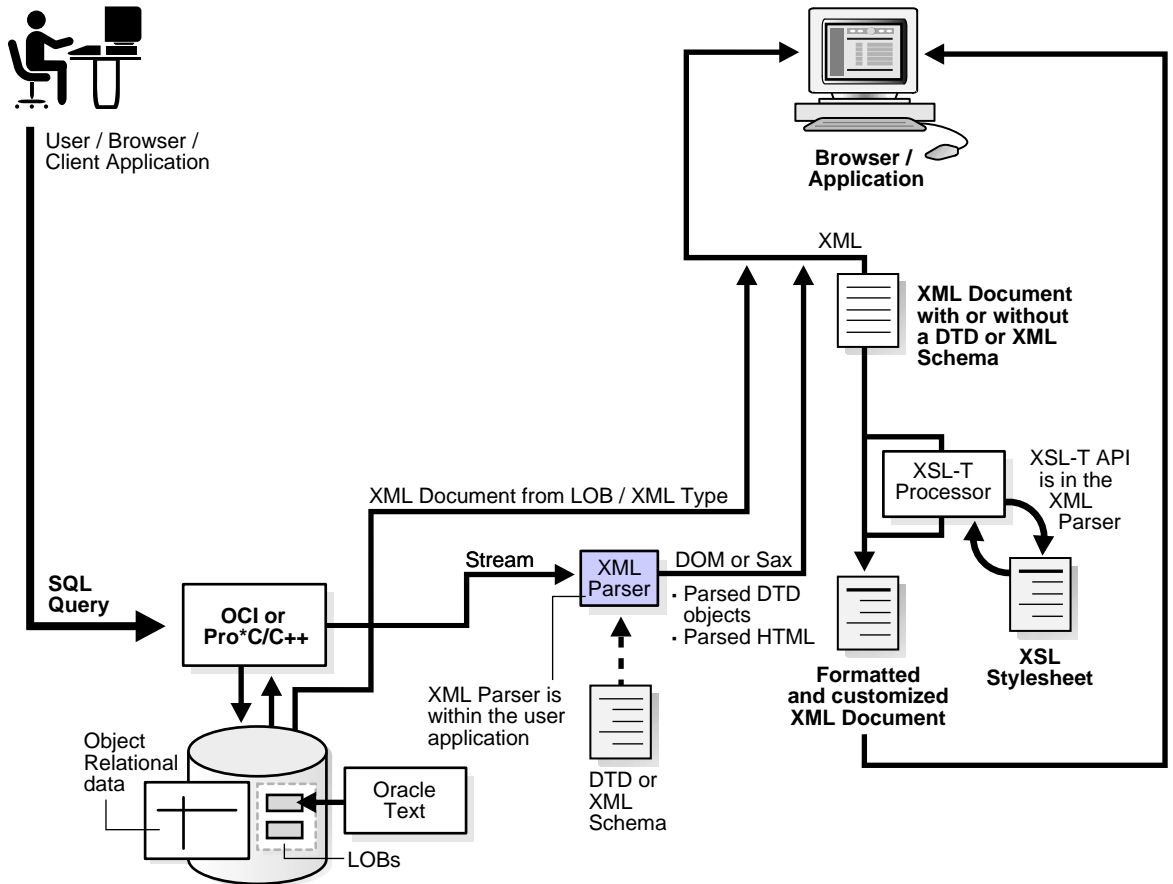
SQL queries can be sent to the database via OCI or as embedded statements in the Pro*C precompiler.

The resulting XML data can be processed in the following ways:

- With the XML Parser
- From the CLOB as an XML document

This XML data is optionally transformed by the XSLT processor, viewed directly by an XML-enabled browser, or sent for further processing to an application or AQ Broker.

Figure 3–8 Generating XML Documents Using XDK for C



Using Oracle XML Components to Generate XML Documents: C++

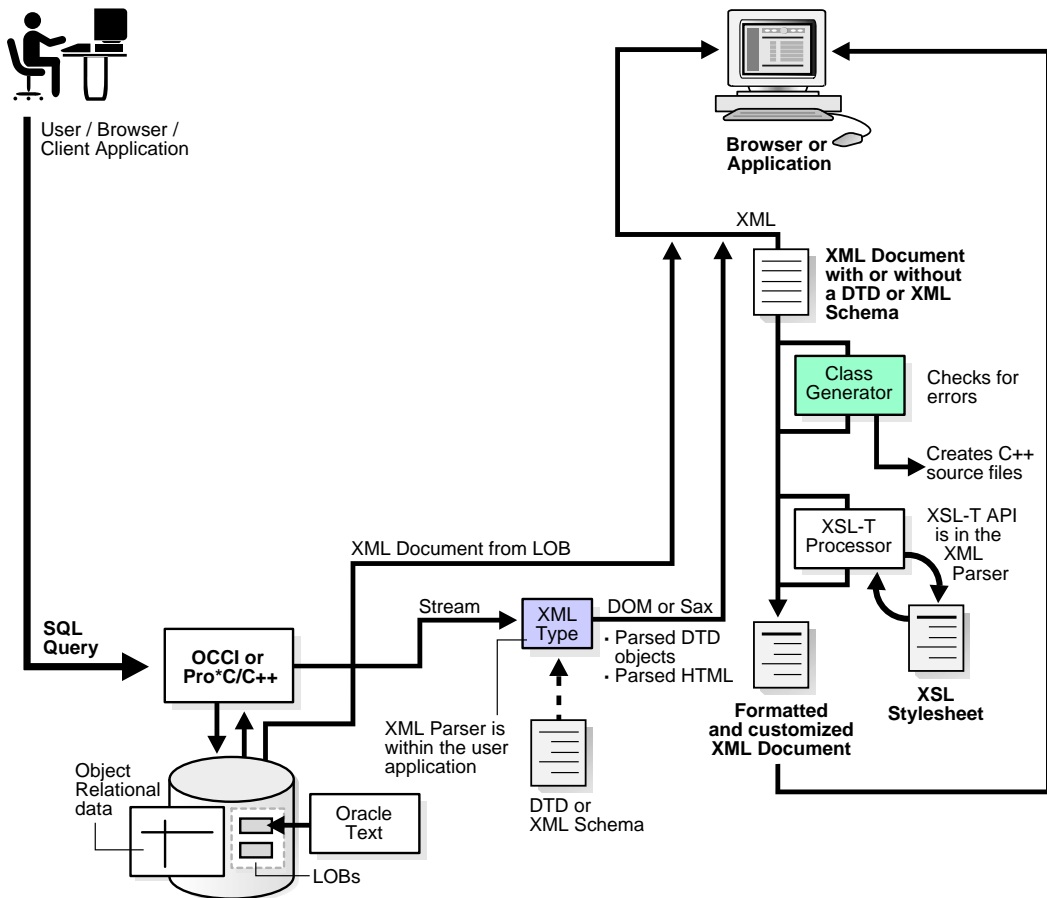
[Figure 3-9](#) shows the Oracle XML components used to generate an XML document. The XDK for C++ components used here are:

- XML Parser for C++, Version 2 including the XSLT
- XML Schema Processor for C++
- XML Class Generator for C++

In the C++ environment, when a user or client or application sends a SQL query, there are two possible ways of processing the query using the XDK for C++:

- Directly by JDBC which then accesses the XML Parser
- Through OCI or Pro*C/C++ Precompiler

Figure 3–9 Generating XML Documents Using XDK for C++

**Oracle9i or other database****XML documents stored:**

- As single object with tags in CLOB or BLOB
- As data distributed untagged across tables
- Via views that combine the documents and data

Using Oracle XML Components to Generate XML Documents: PL/SQL

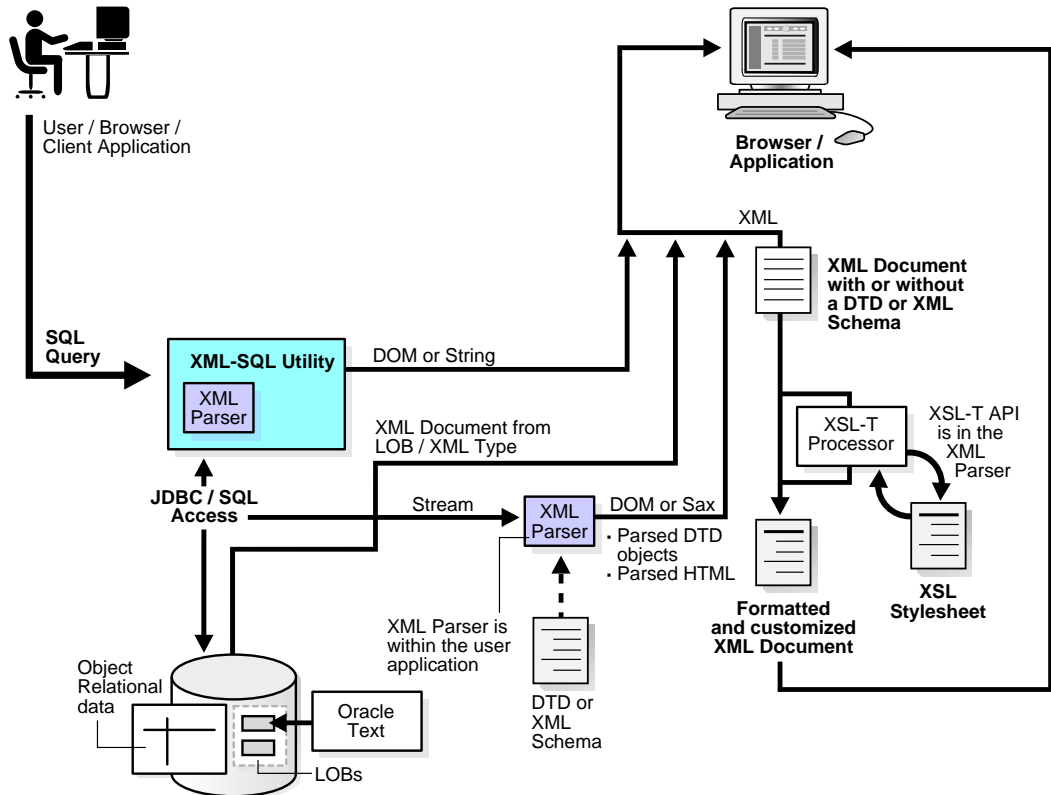
Figure 3–10 shows the XDK for PL/SQL components used to generate an XML document:

- XML Parser for PL/SQL, Version 2 including XSLT
- XML SQL Utility (XSU) for PL/SQL

In the PL/SQL environment, when a user or client or application sends a SQL query, there are two possible ways of processing the query using the Oracle XML components:

- Directly by JDBC which then accesses the XML Parser
- Through XML SQL Utility (XSU)

Figure 3–10 Generating XML Documents Using XDK for PL/SQL



Frequently Asked Questions (FAQs): Oracle XML-Enabled Technology

This section includes general questions about Oracle XML-enabled technology in the following categories:

- [General XDK Questions](#)
- [Portability and XML Support in Older Oracle Releases](#)
- [Browsers that Support XML](#)
- [Standards](#)
- [XML, CLOBs, and BLOBs](#)
- [Maximum FileSizes](#)
- [Inserting XML Data Into Tables](#)
- [XML in the Database: Performance](#)
- [Using XML With Different Languages](#)
- [Further References](#)

There are Frequently Asked Questions at the end of several other chapters in this manual.

General XDK Questions

What XML Components Do I Need to Install?

I am going to develop a small application using XML and Oracle. Here is the scenario: Company A has a central purchasing system with Departments B, C, and D. Company A gets purchase orders in XML format from B, C, and D.

Company A needs to collect all purchase orders and store them in an Oracle database. Then, it has to create another Request for proposal for its preferred vendors in XML. I am writing queries to insert or update into the database. What XML components do I need to install in Oracle?

Answer

Assuming you are using Java, you need the XML Parser and XML SQL Utility. If you are using a Java-based front end to generate the purchase orders, then the XML Class Generator can provide you with the classes you need to populate your purchase orders. Finally, the XSQL Servlet can help you build a Web interface.

Building an XML Application: What Software Is Needed?

I have a CGI-PERL-Oracle7 application on Solaris 2.6 and I want to convert it to XML/XSL-JAVA-Oracle. I know most parts of the technologies, for example, SGML, XML, and JAVA, but I don't know how to start it in Oracle. What software do I need from Oracle? Specifically,

1. Can I use Apache instead of the Oracle Web server? If so, how?
2. How far can I go with Oracle 7.3?
3. Do I still need an XML Parser if all XML was created by my programs?
4. What should be between the Web server and Oracle DB server? XSQL Servlet? Parser? JAVA VM? EJB? CORBA? SQLJ? JDBC? Oracle packages such as UTL_HTTP?

Answer

1. Yes you can. The Apache web server must now interact with Oracle through JDBC or other means. See the XSQL servlet. This is a servlet that can run on any servlet-enabled Web server. This runs on Apache and connects to the database through a JDBC driver to the Oracle database.
2. How far can you go with Oracle 7.3? You can go a long way. The only problem would be that you cannot run any of the Java programs inside the server, that is, you cannot load all the XML tools into the server. But you can connect to the database by downloading the Oracle JDBC utility for Oracle7 and run all the programs as client-side utilities.
3. Do you still need an XML Parser if all XML was created by your programs? That depends on what you intend to do with the generated XML. If your task is just to generate XML and send it out then you might not need it. But if you wanted to generate an XML DOM tree then you would need the parser. Also, you would need it if you have incoming XML documents and you want to parse and store them somewhere. See the XML SQL utility for some help on this issue.
4. What should be between the Web server and Oracle DB server? As explained before in Answer 1, you would need to have a servlet (or CGI) which interacts to Oracle through OCI or JDBC.

DTD to Database Schema

Is there a tool that goes from a DTD to a database schema?

Answer

Currently we do not have a tool to go from a DTD to a database schema as there is no way to specify datatypes until we have XML Schema. With our XML- SQL Utility available on OTN with our other XML components you can generate a DTD from a database schema which can then be entered into the Class Generator. You should try an approach your solution from that angle since a database is involved. Check out our OTN resource including the XML Discussion Forum for further assistance at <http://otn.oracle.com/tech/xml>

Schema Map to XML

My project requires converting master-details data to XML for clients.

1. Is there a best way to design tables and generate XML (flat tables, objects, or collections)?
2. Can I use XML SQL Utilities in Pro*C?
3. Is there a limiting size for generating XML documents from database? Can I use Pro*C to call XSU?

Answer

1. It really depends on what your application calls for. The generalized approach is to use object views and have the schema define the tag structure with database data as the element content.
2. I am not aware of any limits beyond those imposed by the object view and the underlying table structure.

Are There XDK Utilities That Translate From Other Formats to XML?

Are there any utilities in the XDK that translate data from a given format to XML? I know that the XSLT will translate from XML to XML, HTML, or another text-based format. What about the other way around?

Answer

For HTML, you can use utilities like Tidy or JTidy to turn HTML into well-formed HTML that can be transformed using XSLT. For random text formats, you can try utilities like XFlat at <http://www.unidex.com/xflat.htm>.

Can Oracle Generate a Database Schema From a Rational Rose Generated XML File?

It is possible to generate database schema in Oracle using a script with `CREATE TABLE`, from an XML file generated by a Rational Rose design tool?

Answer

All the parser/generator files (such as petal files, XML, and so on) are developed in our project. All the components are designed for reuse, but developed in the context of a larger framework. You have to follow some guidelines, such as modeling in UML, and you must use the base class to get any benefit from our work.

Oracle only generates object types and delivers full object-oriented features such as inheritance in the persistence layer. If you did not need this, the Rational Rose (Petal-File) parser and Oracle packages as the base of the various generators may interest you.

Does Oracle Offer Any Tools to Create and Edit XML Documents?

Does Oracle have any tools for creation (based on DTDs or XML Schema Definition DOM) and editing of XML documents with DTD or Schema validation?

Answer

JDeveloper9i has an integrated XML Schema-driven code editor for working on XMLSchema-based documents such as XML Schemas and XSLT Stylesheets, with tag-insight to help you easily enter the correct elements and attributes as defined by the schema.

See Also: [Chapter 11, "Using JDeveloper to Build Oracle XML Applications"](#)

How Can I Format XML Documents as PDF?

I have been asked to take stored XML docs in v816 and format them as PDF. We are using JDev 3112 as our development environment and the client wants to stick to OAS 4082 on NT if possible. Any suggestions or recommended resources?

Answer

Oracle XSQL Pages v1.0.2 supports integration with Apache FOP 0.14.0 for rendering PDF output from XML/SQL input.

It is possible to format XML into PDF using Formatting Object (FOP). See information on this at: <http://xml.apache.org/fop/> and <http://www.xml.com/pub/rg/75>)

How Do I Load a Large XML Document Into the Database?

Question 1

I have a large (27 MB) data-centric XML document. I could not load it into the database when it was split into relational tables with XML SQL Utility, because the DOM parser failed (memory leak) during the XSLT processor execution. Do you have a workaround for this problem? Should I use SAX Parser? How do I use the XSLT processor and Sax Parser?

Answer 1a

If this is a one time load, or if the XML document you get always has the same tags, then you might consider using the SQL*Loader (direct path). All you have to do is compose a loader control file (see the *Oracle9i Utilities* manual, Chapter 3, for examples). You can use the `enclosed by` option to describe the fields. For example, in the files list you enter something like the following:

```
(empno      number(10)      enclosed by "<empno>" and "</empno>" ,...)
```

Except for the data parsing which has to be done the same regardless of what you are using, the actual loading into the database will be fastest with SQL*Loader (as the direct path writes data straight to data blocks, bypassing the layers in between).

Answer 1b

If the document is 27 MB because it is a very large number of repeating sub-documents, then you can use the sample code that comes in Chapter 14 of the book "Building Oracle XML Applications" by Steve Muench (O'Reilly) to load XML of any size into any number of tables. In Chapter 14, "Advanced XML Loading Techniques", the example builds an XML Loader utility that does what you are looking for.

Question 2

Can SQL*Loader handle nesting? That is, what if you have:

```
...
  <something>
    <price>10.00</price>
```

```

    </something>
...
    ...
        ...
            <somethingelse>
                <price>55.00</price>
            </somethingelse>

```

Is there a way to uniquely identify the two <price> elements?

Answer 2

Not really. The field description in the control file can be nested which is part of the support for object relational columns. The data record to which this maps is of course flat but using all the data-field description features of the SQL*Loader one can get a lot done. For example:

sample.xml

```

<resultset>
  <emp>
    <first>...</first>
    <last>...</last>
    <middle>...</middle>
  <emp>
  <friend>
    <first>...</first>
    <last>...</last>
    <middle>...</middle>
  </friend>
</resultset>

```

samplectl -- field definition part of the SQL Loader control file

```

field list ....
(
  emp COLUMN OBJECT ....
  (
    first      char(30)  enclosed by "<first>" and "</first>",
    last       char(30)  enclosed by "<last>" and "</last>",
    middle     char(30)  enclosed by "<middle>" and "</middle>"
  )
  friend COLUMN OBJECT ....
  (
    first      char(30)  enclosed by "<first>" and "</first>",
    last       char(30)  enclosed by "<last>" and "</last>",

```

```
        middle      char(30)  enclosed by "<middle>" and </middle>"
    )
```

Keep in mind that the `COLUMN OBJECT` field names have to match the ADT column in the database. Also, you will have to use a custom record terminator, otherwise it defaults to `newline` (that is, at every new line it thinks that it has data for a complete database record).

If your XML is more complex and you are trying to extract only select fields, you can use `FILLER` fields to reposition the scanning cursor, which scans from where it has left off towards the end of the record (or for the first field, from the beginning of the record).

The SQL*Loader has a very powerful text parser so you can do a lot of neat tricks. For loading XML when the document is very big, but consistent in its tags, you should consider it.

See Also: [Chapter 2, "Modeling and Design Issues for Oracle XML Applications"](#), ["Loading XML into a Database"](#) on page 2-13, for guidelines on loading XML

Portability and XML Support in Older Oracle Releases

Can I Use Parsers from Different Vendors?

I am currently investigating SAX. I understand that both the Oracle and IBM parsers use DOM and SAX from W3C.

- What is the difference between the parsers from different vendors like Oracle and IBM?
- If I use the Oracle XML Parser now, and for some reason I decide to switch to parser by other vendor, will I have to change my code?

Answer

You will not have to change your code if you stick to SAX interfaces or DOM interfaces for your implementation. That is what the standard interfaces are in place to assist you with.

Is There XML Support in Oracle 8.0.x?

We are currently architecting some of our future systems to run on XML-based interfaces. We are a large Wall Street institution. Our current systems are all running Oracle 8.0.6, and we would like to have some of our XML concepts implemented on the existing systems due to high demand. Are there current or future plans to support XML-based code within the database, or are there any adapters or cartridges that we can use to get by?

Answer

All of our XML Developer's Kit components, including the XML Parser, XSLT Processor, XSQL Servlet, and utilities like the XML SQL Utility all work outside the database against Oracle 8.0.6. However, you will not be able to run XML components inside the database or use Oracle Text (*interMedia*) XML searching, which are both features in Oracle 8i and higher.

Oracle 7.3.4: Data Transfers to Other Vendors Using XML

Question

My company has Oracle release 7.3.4 and my group is thinking of using XML for some data transfers between us and our vendors. From what I could see from this Web site, it looks like we would need to move to Oracle8i or higher in order to do so. Is there any way of leveraging Oracle release 7 to do XML?

Answer

As long as you have the appropriate JDBC 1.1 drivers for 7.3.4 you should be able to use the XML SQL Utility to extract data in XML.

For JDBC drivers, refer to http://otn.oracle.com/tech/java/sqlj_jdbc/ for information about Oracle7 JDBC OCI and JDBC Thin Drivers.

If I Use Versions Prior to Oracle8i Can I use Oracle XML Tools?

1. If I am using an Oracle version lower than Oracle8i, can I supply XML based applications using Oracle XML tools? If yes, then what are the licensing terms in that case?
2. Is Oracle XML technology suitable for creating magtape files where the file is just a string of characters like 'abcdefg ' in a particular format? Is it possible to create a stylesheet that will create these kind of files?

Answer

1. XDKs for Java, C, and C++ can work outside the database, including the XML SQL Utility and XSQL Pages framework. Licensing is the same, free runtime. See OTN for the latest licenses.
2. Yes. Just use `<xsl:output method="text"/>` to output plain text.

Browsers that Support XML

Which Browsers Support XML?

Is there a list of browsers that support XML?

Answer

The following browsers support the display of XML:

- Opera. XML, in version 4.0 and higher
- Citec Doczilla. XML and SGML browser
- Indelv. Will display XML documents only using XSL
- Mozilla Gecko. Supports XML, CSS1, and DOM1
- HP ChaiFarer. Embedded environment that supports XML and CSS1
- ICESoft embedded browser. Supports XML, DOM1, CSS1, and MathML
- Microsoft IE5. Has a full XML parser, IE5.x or higher
- Netscape 5.x or higher

Standards

Are there Advantages of XML Over EDI?

We are considering implementing EDI to communicate requirements with our vendors and customers. I understand that XML is a cheaper alternative for smaller companies. Do you have any information on the advantages of XML over EDI?

Answer

Here are some thoughts on the subject:

- EDI is a difficult technology: EDI allows machine-to-machine communication in a format that developers cannot easily read and understand.
- EDI messages are very difficult to debug. XML documents are readable and easier to edit.
- EDI is not flexible: it is very hard to add a new trading partner as part of an existing system, each new trading partner requires its own mapping. XML is extremely flexible with the ability to add new tags on demand and to transform an XML document into another XML document, for example, to map two different formats of purchase order numbers.
- EDI is expensive: developer training costs are high, and deployment of EDI requires very powerful servers that need a specialized network. (EDI runs on VANs, which are expensive). XML works with inexpensive Web servers over existing internet connections.

The next question then becomes: is XML going to replace EDI? Probably not. The technologies will likely coexist, at least for a while. Large companies with an existing investment in EDI will probably use XML as a way to extend their EDI implementation, which raises a new question of XML and EDI integration.

XML is a compelling approach for smaller organizations, and for applications where EDI is inflexible.

What B2B Standards and Development Tools Does Oracle Support?

What B2B XML standards (such as ebXML, cxml, and BizTalk) does Oracle support? What tools does Oracle offer to create B2B exchanges?

Answer

Oracle participates in several B2B standards organizations:

- OBI (Open Buying on the Internet)
- ebXML (Electronic Business XML)
- RosettaNet (E-Commerce for Supply Chain in IT Industry)
- OFX (Open Financial Exchange for Electronic Bill Presentment and Payment)

For B2B exchanges, Oracle provides several alternatives depending on customer needs, such as the following:

- Oracle Exchange delivers an out-of-the-box solution for implementing electronic marketplaces

- Oracle Integration Server (and primarily Message Broker) for in-house implementations
- Oracle Gateways for exchanges at data level
- Oracle XML Gateway to transfer XML-based messages from our e-business suite.

Oracle Internet Platform provides an integrated and solid platform for B2B exchanges.

What is Oracle Corporation's Direction Regarding XML?

What is Oracle Corporation's direction regarding XML?

Answer

Oracle Corporation's XML strategy is to use XML in ways that exploit all of the benefits of the current Oracle technology stack. Today you can combine Oracle XML components with the Oracle8i (or higher) database and Advanced Queueing (AQ) to achieve conflict resolution, transaction verification, and so on. Oracle is working to make future releases more seamless for these functions, as well as for functions such as distributed two phase commit transactions.

XML data is stored either object-relational tables or views, or as CLOBs. XML transactions are transactions with one of these datatypes and are handled using the standard Oracle mechanisms, including rollback segments, locking, and logging.

From Oracle, Oracle supports sending XML payloads using AQ. This involves making XML queryable from SQL.

Oracle is active in all XML standards initiatives, including W3C XML Working Groups, Java Extensions for XML, Open Applications Group, and `XML.org` for developing and registering specific XML schemas.

XML Query

Oracle is participating in the W3C Working Group for XML Query. Oracle is considering plans to implement a language that allows querying XML data, such as in the XQL proposal. While XSLT provides static XML transformation features, a query language will add data query flexibility similar to what SQL does for relational data.

Oracle has representatives participating actively in the following W3C Working Groups related to XML/XSL: XML Schema, XML Query, XSL, XLink/XPointer, XML Infoset, DOM, and XML Core.

Are There Standard DTDs that We Can Use for Orders, Shipments, and So On?

We have implemented Oracle8i and the XDK. Where can we find basic, standard DTDs to build on for orders, shipments, and acknowledgements?

Answer

A good place to start would be this Web site: <http://www.xml.org> which is being set up for that purpose.

XML, CLOBs, and BLOBs

Is There Support for XML Messages in BLOBs?

Is there any support for XML messages enclosing BLOBs, or I should do it on an application level by encoding my binary objects in a suitable text format such as UUENCODE with a MIME wrapper?

Answer

XML requires all characters to be interpreted, therefore there is no provision for including raw binary data in an XML document. That being said, you could UUENCODE the data and include it in a CDATA section. The limitation on the encoding technique is to be sure it only produces legal characters for a CDATA section.

Maximum FileSizes

What is the Maximum XML File Size When Stored in CLOBs?

If we store XML files as CLOBs in the Oracle database, what is the maximum file size?

Answer

The maximum file size is 2 GB. See the *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for more information on LOBs and CLOBs.

XML File Size Limitations

Are there any limitations in the size of an XML file?

Answer

There are no XML limitations to an XML file size.

Maximum Size for an XML Document

1. Is there a maximum size for an XML document to provide data for PL/SQL (or SQL) across tables, provided that no CLOBs are used?
2. What is the maximum size of XML document generated from Oracle to an XML document?

Answer

1. The size limit should be what can be inserted into an object view.
2. The size limit should be what can be retrieved from an object view.

Inserting XML Data Into Tables

What Do I Need to Insert Data Into Tables Using XML?

To select data for display and insert data to tables by XML what software do I need?
We are using Oracle8i on Solaris.

Answer

You need the following software:

- XML SQL Utility
- XML Parser for Java,V2
- JDBC driver
- JDK

The first three can be obtained from Oracle. The fourth can be obtained from Sun Microsystems. If you want to perform the tasks from a browser, you will also need the following:

- A Java compliant Web server
- XSQL Servlet

XML in the Database: Performance

Where Can I Find Information about the Performance of XML and Oracle?

Is there a whitepaper that discusses the performance of XML and Oracle?

Answer

Currently, we do not have any official performance analyses due to the lack of a performance standard or benchmark for XML products.

How Can I Speed Up the Record Retrieval in XML Documents?

I have a database with millions of records. I give a query based on some 4/5 parameters, and retrieve the records corresponding to that, I have added indexes in the database for faster retrieval of the same, but since the number of records returned is quite high and I planned to put a previous and next link to show only 10 records at a time, I had to get the `count(*)` of the number of records that match.

Since there are so many records, and `count(*)` does not consider index, it takes nearly 20-30 seconds for the retrieved list to be seen on the browser window. If I remove that `count(*)`, the retrieval is quite fast, but then there is no previous and next as I had linked them to `count(*)`.

Answer

I presume you are referring on a faster way to retrieve XML documents. The solution is to use SAX interface instead of DOM.

Make sure to select the `COUNT(*)` of an indexed column (the more selective the index the better), this way the optimizer can satisfy the count query with a few I/Os of the index blocks instead of a full-table scan.

Using XML With Different Languages

My application requires communication with outside entities that may have a totally different language system. If I need to put information in other languages (for instance, Chinese) into XML, do I need to treat and process them differently? For example, do I need to care which encoding they use, or would the parser be able to recognize it? Would there be any problems when dealing with the database?

Answer

XML inherently supports multiple languages in a single document. Each entity can use a different encoding from the others; that is, you could add a Chinese entity encoded in a Chinese encoding to the rest of the document. You could also treat all portions uniformly, regardless of the language used, by encoding in Unicode. Using the former, you must have an encoding declaration in the XML text declaration.

Oracle XML Parsers are designed to be able to handle most external entities and recognizes a wide range of encodings, including most widely used ones from all over the world.

The database should support all the languages you are going to use on XML. Chinese character sets like ZHS16GBK and ZHT16BIG5 are a superset of ASCII so you may be able to do with one of them to serve for English and Chinese, but you may want to use Unicode to use more languages.

Further References

Other XML Frequently Asked Questions

Here is another XML Frequently Asked Question site of interest:

- <http://www.ucc.ie/xml/>

Recommended XML and XSL Books

Can you recommend a good XML or XSL book?

Answer

- A publisher group by the name of WROX has a number of helpful books. One of these, *XML Design and Implementation* by Paul Spencer covers XML, XSL and development well.
- *Building Oracle XML Applications* by Steve Muench (published by O'Reilly) See <http://www.oreilly.com/catalog/orxmlapp/>
- *The XML Bible*. Although I do not have this book, my impression that it is good one on XML and XSL. I read the updated chapter 14 from: <http://metalab.unc.edu/xml/books/bible/> and it gave me a good understanding of XSLT. Downloading this chapter is free so you can get a good impression.

- *Oracle XML Handbook* by the Oracle XML Product Development Team
<http://www.osborne.com/oracle/>

Using XSL and XSLT

This chapter contains the following sections:

- [Introducing XSL](#)
- [XSL Transformation \(XSLT\)](#)
- [XML Path Language \(XPath\)](#)
- [CSS Versus XSL](#)
- [XSL References](#)
- [Frequently Asked Questions: XSL and XSLT](#)

Introducing XSL

XML documents have structure but no format. Extensible Stylesheet Language (XSL) adds formatting to XML documents.

XSL provides a way of displaying XML semantics. It can map XML elements into other formatting languages such as HTML.

The W3C XSL Specification

The W3C is developing the XSL specification as part of its Style Sheets Activity. XSL has document manipulation capabilities beyond styling. It is a stylesheet language for XML.

The July 1999 W3C XSL specification, was split into two separate documents:

- XSL syntax and semantics
- How to use XSL to apply style sheets to transform one document into another

The formatting objects used in XSL are based on prior work on Cascading Style Sheets (CSS) and the Document Style Semantics & Specification Language (DSSSL). XSL is designed to be easier to use than DSSSL.

Capabilities provided by XSL as defined in the proposal enable the following functionality:

- Formatting of source elements based on ancestry and descendency, position, and uniqueness
- The creation of formatting constructs including generated text and graphics
- The definition of reusable formatting macros
- Writing-direction independent stylesheets
- An extensible set of formatting objects.

XSL Specification Proposal

The XSL specification defines XSL as a language for expressing stylesheets. Given a class of arbitrarily structured XML documents or data files, designers use an XSL stylesheet to express their intentions about how that structured content should be presented; that is, how the source content should be styled, laid out, and paginated in a presentation medium, such as a window in a Web browser or a hand-held device, or a set of physical pages in a catalog, report, pamphlet, or book. Formatting is enabled by including formatting semantics in the result tree.

Formatting semantics are expressed in terms of a catalog of classes of formatting objects. The nodes of the result tree are formatting objects. The classes of formatting objects denote typographic abstractions such as page, paragraph, table, and so forth.

Finer control over the presentation of these abstractions is provided by a set of formatting properties, such as those controlling indents, word and letter spacing, and widow, orphan, and hyphenation control. In XSL, the classes of formatting objects and formatting properties provide the vocabulary for expressing presentation intent.

An implementation is not mandated to provide these as separate processes. Furthermore, implementations are free to process the source document in any way that produces the same result as if it were processed using the conceptual XSL processing model.

Namespaces in XML

A namespace is a unique identifier or name. This is needed because XML documents can be authored separately with different DTDs or XML Schemas. Namespaces prevent conflicts in markup tags by identifying which DTD or XML Schema a tag comes from. Namespaces link an XML element to a specific DTD or XML Schema.

Before you can use a namespace marker such as `rml:`, `xhtml:`, or `xsl:`, you must identify it using the namespace indicator, `xmlns` as shown in the next paragraph.

See Also: <http://w3.org/TR/REC-xml-names>

XSL Stylesheet Architecture

The XSL stylesheets must include the following syntax:

- Start tag stating the stylesheet, such as `<xsl:stylesheet2>`
- Namespace indicator, such as `xmlns:xsl="http://www.w3.org/TR/WD-xsl"` for an XSL namespace indicator and `xmlns:fo="http://www.w3.org/TR/WD-xsl/FO"` for a formatting object namespace indicator
- Template rules including font families and weight, colors, and breaks. The templates have instructions that control the element and element values
- End of stylesheet declaration, `</xsl:stylesheet2>`

XSL Transformation (XSLT)

XSLT is designed to be used as part of XSL. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.

Meanwhile the second part is concerned with the XSL formatting objects, their attributes, and how they can be combined.

See Also: [Chapter 20, "Using XML Parser for Java"](#)

XSLT 1.1 Specification

The W3C Working Group on XSL has just released a document describing the requirements for the XSLT 1.1 specification. The primary goal of the XSLT 1.1 specification is to improve stylesheet portability. The new draft is available at <http://www.w3.org/TR/xslt11req>

In addition to supporting user-derocessors have exploited the XSLT 1.0 extension mechanism to provide additional built-in transformation functionality. As useful built-in extensions have emerged, users have embraced them and have begun to rely on them.

However the benefits of these extensions come at the price of portability. Since XSLT 1.0 provides no details or guidance on the implementation of extensions, today any user-written or built-in extensions are inevitably tied to a single XSLT processor.

Goal 1. Improve Stylesheet Portability

The primary goal of the XSLT 1.1 specification is to improve stylesheet portability. This goal will be achieved by standardizing the mechanism for implementing extension functions, and by including in the core XSLT specification two of the built-in extensions that many existing vendors XSLT processors have added due to user demand:

- Support for multiple output documents from a transformation
- Support for converting a result tree fragment to a nodeset for further processing
By standardizing these extension-related aspects which multiple vendor implementations already provide, the ability to create stylesheets that work across multiple XSLT processors should improve dramatically.

Goal 2. Support the New XML Specification

A secondary goal of the XSLT 1.1 specification is to support the new XML base specification.

The XSLT 1.1 specification proposal provides the requirements that will achieve these goals. The working group has decided to limit the scope of XSLT 1.1 to the standardization of features already implemented in several XSLT 1.0 processors, and concentrate first on standardizing the implementation of extension functions.

Standardization of extension elements and support for new XML Schema data type aware facilities are planned for XSLT 2.0.

XML Path Language (XPath)

A separate, related specification is published as the XML Path Language (XPath) Version 1.0. XPath is a language for addressing parts of an XML document, essential for cases where you want to specify exactly which parts of a document are to be transformed by XSL. For example, XPath lets you select all paragraphs belonging to the chapter element, or select the elements called special notes. XPath is designed to be used by both XSLT and XPointer. XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL transformations and XPointer.

CSS Versus XSL

W3C is working to ensure that interoperable implementations of the formatting model is available.

Cascading Stylesheets (CSS)

Cascading Stylesheets (CSS) can be used to style HTML documents. CSS were developed by the W3C Style Working Group. CSS2 is a style sheet language that allows authors and users to attach styles (for example, fonts, spacing, or aural cues) to structured documents, such as HTML documents and XML applications.

By separating the presentation style of documents from the content of documents, CSS2 simplifies Web authoring and site maintenance.

XSL

XSL, on the other hand, is able to transform documents. For example, XSL can be used to transform XML data into HTML/CSS documents on the Web server. This

way, the two languages complement each other and can be used together. Both languages can be used to style XML documents. CSS and XSL will use the same underlying formatting model and designers will therefore have access to the same formatting features in both languages.

The model used by XSL for rendering documents on the screen builds on years of work on a complex ISO-standard style language called DSSSL. Aimed mainly at complex documentation projects, XSL also has many uses in automatic generation of tables of contents, indexes, reports, and other more complex publishing tasks.

XSL References

Examples on using XSL can be found throughout this manual. In particular, refer to the following chapters in *Oracle9i Case Studies - XML Applications*:

- "Customizing Content with XML: Dynamic News Application"
- "OracleAS Wireless Edition and XML"
- "Customizing Presentation with XML and XSQL: Flight Finder"

See Also:

- <http://www.mulberrytech.com/xsl/xsl-list/>
- http://www.builder.com/Authoring/XmlSpot/?tag=st.cn.sr1.ssr.bl_xml

Frequently Asked Questions: XSL and XSLT

How Do I Write an IF Statement in XSL That Tests for Values Within Tags?

What is the syntax to compare not an element but the value of the element? So far, the documentation I have read tests for tags but not values within the tags. Here is a portion of my XSL document:

```
<xsl:template match="EmployeeList">
  <xsl:for-each select="employee">
    <xsl:value-of select="name"/>
    <xsl:value-of select="sal"/>
  </xsl:for-each>
</xsl:template>
```

I want to construct an IF statement that will display the information of employees with salaries greater than 5000 in red. How do I insert the value of `sal` in the IF statement?

Answer

Here is the IF statement:

```
<xsl:if expr="this.nodeTypeValue == 'INIZIATIVE'">
    .....
</xsl:if>
```

In an XSL Document, How Can We Select Specific Attributes?

We are merging an XML document with its XSL stylesheet. However the child attributes are not being returned when we use syntax of type:

```
<xsl:value-of select="Foo/Bar"/>
```

in the XSL document. Why not? This seems to work fine in other XML parsers.

Answer

The XPath expression, `Foo/Bar`, is only designed to select the value of the `<Bar>` element contained in the `<Foo>` element. It will return the concatenation of all text nodes in the nested content of that `<Bar>` element, but certainly is not designed to select any text values in attributes.

For this, you would need the syntax: `Foo/Bar/@SomeAttr`

to select one attribute and...

```
Foo/Bar/@*
```

to select all the attributes of `<Bar>`

When Converting XML to HTML, Why Do I get "Unexpected EOF"?

I am trying to render a simple XML document to an HTML form, using the following XML and XSLT. The transformation fails with the message "Unexpected EOF" using the `XSLSample.java` provided with the XML parser for Java V2. When I remove the `<td></td>` from the transformation (which contains the XPath expression of the type `{ELEMENT}`), the transformation is fine.

Here is the XML:

```
<ROWSET>
```

```
<ROW>
  <ELEM0>Al</ELEM0>
  <ELEM1>Gore</ELEM1>
  <ELEM2></ELEM2>
  <ELEM3></ELEM3>
  <ELEM4></ELEM4>
</ROW>
.....
```

Here is the XSLT:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
  <head>
    <title>Value Upload</title>
  </head>
  <body bgcolor="#FFFFFF">
    <form method="post" action="">
      <xsl:for-each select="ROWSET">
        <table border="1" cellspacing="0" cellpadding="0">
          <xsl:for-each select="ROW">
            <tr>
              <td><input type="text" name="elem0" value="{ELEM0}" size="10"
                maxlength="20"></td>
              <td><input type="text" name="elem1" value="{ELEM1}" size="10"
                maxlength="20"></td>
              ...
            </xsl:for-each>
          </form>
        </body>
      </html>
    </xsl:template>
  </xsl:stylesheet>
```

Answer

You need to put a slash (/) for the input element, as follows:

```
<td> <input xxxx /> </td>
```

Whitespace: Why are my Resulting Values Multiplied by 2?

Is there a syntax error in the following code?

Here is `djia.xml`:


```
<?xml version="1.0" encoding="Shift_JIS"?>
<?xml-stylesheet type="text/xsl" href="djia.xsl"?>
<djia>
  <company>ALCOA</company>
  <company>ExxonMobil</company>
  <company>McDonalds</company>
  <company>American Express</company>
</djia>
```

Here is djia.xsl:

```
<?xml version="1.0" encoding="Shift_JIS"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"version="1.0">
<xsl:output method="xml" encoding="Shift_JIS"/>
<xsl:template match="/djia">
<page>
  <xsl:apply-templates/>
</page>
</xsl:template>
<xsl:template match="company">
  <xsl:value-of select="current()"/>
  <xsl:value-of select="position()"/>:
  <xsl:if test="current()=../company[last()]"> last one!
</xsl:if>
</xsl:template>
</xsl:stylesheet>
```

yields the following:

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<page>ALCOA2: ExxonMobil4: McDonalds6: American Express8: last one!</page>
```

Why the resulting numbers are multiplied by 2?

Answer

The answer is whitespace. When your /djia template does `<xsl:apply-templates/>` it selects all child nodes of `<djia>`. Since your `djia.xml` is nicely indented, that means that child nodes of `<djia>` are:

1. TextNode containing CR + spaces to make next element look indented
2. `<company>` (with value ALCOA)
3. TextNode containing CR + spaces to make next element look indented
4. `<company>` (with value ExxonMobil)

5. TextNode containing CR + spaces to make next element look indented
6. <company> (with value McDonalds)
7. TextNode containing CR + spaces to make next element look indented
8. <company> (with value American Express)
9. TextNode containing CR + spaces to put </djia> on next line.

So as the XSLT processor is processing this current node list, the `position()` function is the position in the current node list, which are 2, 4, 6, 8 for the <company> element.

You should be able to fix the problem by adding a top level:

```
<xsl:strip-space elements="*" />
```

However, a bug in XDK for Java currently prevents this from working correctly. One workaround is to use:

```
<xsl:apply-templates select="company" />
```

instead of only:

```
<xsl:apply-templates />
```

How Can I Specify a NULL Indicator in XSL?

I want my XSLT to output `<mytag null="yes" />` when my corresponding source XML is `<mytag />` or `<mytag NULL="YES" />`. How do I specify that within my XSLT?

Answer

Use the following syntax:

```
<xsl:template match="mytag">
  <!-- If there are no child nodes -->
  <xsl:if test="not(node())">
    <mytag null="yes" />
  </xsl:if>
</xsl:template>
```

How Can Transfer Tag Names in XSLT?

I need to use XSLT to change my XML code from:

```
<REF_STATUS>
...
</REF_STATUS>
```

to:

```
<REF index="STATUS">
...
</REF>
```

and similar code for REF_VATCODE and REF_USFLG. Here is the first attempt I wrote, which works:

```
<!-- fix REF_STATUS nodes -->
<xsl:template priority="1" match="REF_STATUS">
  <xsl:element name="REF">
    <xsl:attribute name="index">STATUS</xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>

<!-- fix REF_USFLG nodes -->
<xsl:template priority="1" match="REF_USFLG">
  <xsl:element name="REF">
    <xsl:attribute name="index">USFLG</xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>

<!-- fix REF_VATCODE nodes -->
<xsl:template priority="1" match="REF_VATCODE">
  <xsl:element name="REF">
    <xsl:attribute name="index">VATCODE</xsl:attribute>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

There are three tag names all beginning with REF_, that are changed into the REF tagname with an index attribute equal to the remainder of the original tag name. I'd like to make one rule which matches all of these and does the correct transformation. Here is one attempt:

```
<xsl:template priority="1" match="starts-with(local-name(),'REF_')">
  <xsl:element name="REF">
    <xsl:attribute name="index">
      <xsl:value-of select="substring-after(local-name(),'REF_')"/>
    </xsl:attribute>
  </xsl:element>
</xsl:template>
```

```
</xsl:attribute>
  <xsl:apply-templates/>
</xsl:element>
</xsl:template>
```

Unfortunately, I get this error message:

Error occurred while processing eName.xsl: XSL-1013: Error in expression: 'starts-with(local-name(), 'REF_').'.

What is wrong with the above expression?

Answer

The following works for me:

Note the `match="starts-with(..)"` is illegal because it is not a valid match pattern. You will need:

```
match="*[starts-with(local-name(), 'REF_')]"
```

as shown below:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- Identity Transform -->
  <xsl:template match="node()|@">
    <!-- Copy the current node -->
    <xsl:copy>
      <!-- Including any attributes it has and any child nodes -->
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template priority="2" match="*[starts-with(local-name(), 'REF_')]">
    <REF index="{substring-after(local-name(), 'REF_')}">
      <xsl:apply-templates/>
    </REF>
  </xsl:template>
</xsl:stylesheet>
```

This transforms a document like:

```
<foo>
  <bar>
    <REF_STATUS>
      <baz/>
    </REF_STATUS>
```

```

    <zoo>
      <REF_USFLG>
        <boo/>
      </REF_USFLG>
    </zoo>
  </bar>
</foo>

```

into the result:

```

<foo>
  <bar>
    <REF index="STATUS">
      <baz/>
    </REF>
    <zoo>
      <REF index="USFLG">
        <boo/>
      </REF>
    </zoo>
  </bar>
</foo>

```

How Do I Convert A String to a Nodelist in XSL?

Question 1

The XML we receive is wrapped with extra code using CDATA notation. My XSL is not picking up the elements in the CDATA section. What do I need to do?

Answer 1

Inside a `<![CDATA[]]>` are not elements and attributes for querying with XPath. They are just literal characters (angle brackets, names, and quotes) that look like elements and attributes, but are not in the infoset tree as separate nodes.

Inside a CDATA there is just a single text node. XSL will not pick up elements in the CDATA. The best you can do is:

- Match on string content of the CDATA
- Programmatically parse the document and programmatically replace the CDATA node by the result of parsing the CDATA node's content as an XML document.

- Extract and parse the string-content of the CDATA and process that with XSLT

Question 2

In one of your examples, I found an XSL file, `toolbar.xsl`, that does what appears to be converting strings to a nodeset by doing the following XSL:

```
<xsl:variable name="barns"select="my:nodeset($bar)"/>
<xsl:stylesheet version="1.0"xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:my="http://www.oracle.com/XSL/Transform/java/oracle.xml.parser.v2.Extensions"exclude-result-prefixes="my">
  <xsl:template match="/">
    <xsl:call-template name="toolbar">
      <xsl:with-param name="bar">
        <toolbar>
          <button name="xxx" url="www.oracle.com"/>
        </toolbar>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:template>
  ...
```

Is my observation correct? I have extracted the CDATA section into a variable, but I think I need to convert it to a nodeset. When I tried it using `AsyncTransformSample.java` in `TransView` bean, I get the error:

```
XSL-1045: Extension function error: Class not found
'oracle.xml.parser.v2.Extensions'
```

Is this part of the standard packages or do I need to import it. The import statement:

```
import oracle.xml.parser.v2.*;
```

is already in `AsyncTransformSample`.

Answer 2

No. The `ora:node-set()` converts result tree fragments to nodesets, not strings to nodesets. To convert strings to nodesets you have to parse the string. XSLT does not have a built-in `parse-string()` function, so we can build one as a Java extension function. See Chapter 16 in *Developing Oracle XML Applications* by Steve Muench (O'Reilly) for details on developing and debugging Java XSLT extension functions.

Here is an example Java class that parses a string and returns a nodeset containing the root node of the parsed XML document in a string. If there is an error during parsing, it returns an empty nodeset.

```
import org.w3c.dom.*;
import org.xml.sax.SAXException;
import oracle.xml.parser.v2.*;
import java.io.StringReader;
public class Util {
    public static NodeList parse (String s) {
        // Create a new parser
        DOMParser d = new DOMParser();
        try {
            // Parse the string into an in-memory DOM tree
            d.parse( new StringReader(s) );
            // Return a node list containing the root node
            return ((XMLDocument)d.getDocument()).selectNodes("/");
        }
        catch (Exception e) {
            // Return an empty nodelist in case of an error.
            return (new XMLDocument()).getChildNodes();
        }
    }
}
```

Here is a sample `message.xml` file that simulates the scenario you are in with some XML in the body of an XML document enclosed in a CDATA section.

```
<message>
  <from>Steve</from>
  <to>Albee</to>
  <body><![CDATA[
    <order id="101">
      <item id="12" qty="10"/>
      <item id="13" qty="3"/>
    </order>
  ]]></body>
</message>
```

Here is a sample stylesheet that processes the `<message>` document, parses and captures the subdocument (that is encoded as a CDATA text node in the `<body>`) in an XSL variable, and then uses `<xsl:for-each>` to select information out of the `$body` variable containing the now-parsed message body. Here we just print out the identifiers of the `<order>`, but this will give you a general idea.

```

<xsl:stylesheet version="1.0"
  xmlns:util="http://www.oracle.com/XSL/Transform/java/Util"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!--
  | Above, we've associated the "util" namespace prefix
  | with the appropriate namespace URI that maps to
  | the "Util" class. The Util.java class is not in any
  | package, otherwise the URI would have looked like
  | http://www.oracle.com/XSL/Transform/java/my.pkg.Util
  +-->

  <xsl:template match="message">
    <!--
    | Use the parse() function in the util namespace
    | to parse the string value of the <body> child
    | element of the current <message> element, and
    | return the root node of the document
    +-->
    <xsl:variable name="body" select="util:parse(body)"/>
    <xsl:text>Items Ordered</xsl:text><xsl:text>&#xa;</xsl:text>
    <xsl:for-each select="$body/order/item">
      <xsl:value-of select="@id"/><xsl:text>&#xa;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>

```

In XSL, How Can I Correctly Convert an XML Document Tag to a Link in HTML?

I have a question about XSL. My XML document is similar to the following:

```

<ROW num="1">
  <TITLE>New Java Classes</TITLE>
  <URL>/products/intermedia/</URL>
  <DESCRIPTION>&#60;a href="/products/intermedia/">Java classes for
  Servlets and JSPs&#60;/a>are available.
  </DESCRIPTION>
</ROW>

```

When I use XSL to display the XML document in HTML, the description is not displaying as a link eventhough I am specifying it as "" in XML.

My XSL file is:

```

<xsl:template>
  <P><FONT face="arial" size="4"><B>

```



```

<xsl:value-of disable-output-escaping="yes" select="TITLE" />
</B> </FONT><BR></BR><FONT size="2">
<xsl:value-of disable-output-escaping="yes" select="DESCRIPTION" /></FONT>
</P>
</xsl:template>

```

Answer

You can simply build the `<a>` tag in your XSL transform. Do something like this:

```

<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html" />
<xsl:template match="/">
  <xsl:for-each select="rowset">
    <table border="1">
      <th>Category</th>
      <th>ID</th>
      <th>Title</th>
      <th>Thumbnail</th>
      <xsl:for-each select="row">
        <tr>
          <td><xsl:value-of select="category" /> </td>
          <td><xsl:value-of select="id" /> </td>
          <td><a href="Present.jsp?page=PRES_VIEW_SINGLE&amp;id={id}"><xsl:value-of
select="title" /> </a></td>
          <td></td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>

```

Am I Using the Correct XSL Headers for my WML Transformation?

I am using `oracle.xml.async.XSLTransformer` included in XDK for Java v2 to perform an XSL transformation on an XML document. I need a WML output. My stylesheet contains the following code:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" omit-xml-declaration="no" encoding="ISO-8859-1" />
<xsl:output doctype-system="http://www.wapforum.org/DTD/wml_1.1.xml"
doctype-public="-//WAPFORUM//DTD WML 1.1/EN"/>

```

...

When I check the transformation using a servlet, I get the following error in my WAP emulator:

```
"Received HTTP status: 502 - WML Encoding Error, 1:com.sun.xml.parser/P-076
Malformed UTF-8 char"
```

Is an XML encoding declaration missing? In fact, the WML generated is not including any XML header information. The output starts like this:

```
<wml>
<card id="gastronomia" title="Mis direcciones de gastronomia"><p>Mis
direcciones de gastronomia</p>
```

...

How do I get the transformer to output the XML header:

```
"<?xml version="1.0" encoding="ISO-8859-1"?>"
```

Answer

Use `oracle.xml.parser.v2.XSLProcessor`. Also ensure your stylesheet has:

```
<xsl:output method="xml"/>
```

just inside the `<xsl:stylesheet>`, and outside of any `<xsl:template>`

Also ensure that you're using the following API:

```
processXSL(stylesheet, source, printwriter)
```

In an XSL Transformation, How Do I Ensure that the DTD File Can be Located?

My BC4J source XML file has the following line that refers to the DTD:

```
<!DOCTYPE ViewObject SYSTEM "jbo_03_01.dtd">
```

When transforming the file, this line results in an error, saying it cannot find `jbo_03_01.dtd`. The DTD file is in my classpath.

Answer

There are two solutions to this.

- Extract `jbo_03_01.dtd` from `jbomt.zip` and put it in the same directory as your VO files. This is complicated if you have VO files at several different directory levels.

- Use the trick that BC4J itself uses when parsing its own XML metadata files when parsing in your program:

```
DOMParser d = new DOMParser();
    // read DTD as a resource from the classpath
    InputStream is = ...getResourceAsStream("/jbo_03_01.dtd");
    d.parseDTD( is );
    DTD dtd = d.getDoctype();
    d.setDoctype( dtd ); // set and cache the DTD to use.

// Now, subsequences calls to d.parse() will

    // use the cached version of jbo_03_01.dtd
```

Then transform the result using XSLStylesheet and
XSLProcessor.process(style,source,printwriter).

In XSL, How to Prevent the Namespace Definition from Being Repeated

My second question relates to namespaces. I have the following piece of code in my stylesheet:

```
<xsl:attribute name="data:text">
    <xsl:value-of select="@Name"/>@ipet:dataBindingObject
</xsl:attribute>
```

At the top of my stylesheet, I have defined the marlin namespace:

```
xmlns:data="http://xxx.us.yyy.com/cabo/marlin"
```

In the resulting XML file (the marlin UIX file), the namespace definition is repeated for each element:

```
<messageTextInput id="Status" name="Status" prompt="Status"
required="yes"xmlns:data="http://xxx.us.yyy.com/cabo/marlin"
data:text="Status@ipet:dataBindingObject" rows="1" maximumLength="3"
columns="3"/>
```

Answer

Try defining the data namespace prefix on the document element in your XSLT root template. If it is defined at a higher level in the result tree we may notice that and not output it on each lower level element.

JDeveloper9i has virtual Virtual Objects (VOs) that expose the metadata of aVO kind of like the database X\$ views. This means that you could use the normal

`VO.writeXML()` method against one of these virtual metadata views to perform operations like I think you are trying to do to render a data-driven output based on the structure of a given VO.

How Do I Pass a Parameter from a Java Program to an XSL Stylesheet?

Is there a way to pass a parameter from a Java program to an XSLT stylesheet using Oracle XSL processor? The XSLT standard states that "...XSLT does not define the mechanism by which parameters are passed to the stylesheet." (see <http://www.w3.org/TR/xslt#variable-values>). This is possible, but is a vendor-dependant implementation. However, none of the XSL constructors in the `OracleXMLprocessor` seems to allow for this.

We need to pass in an integer to a stylesheet and use the `xsl:position()` function to extract a document fragment from an XML doc. For example:

```
<xsl:templatematch="ROW">
<xsl:if test="position()=1">
  SELECT DISTINCT sp.site_datatype_id
  FROM ref_hm_site_pcode sp
  WHERE sp.hm_site_code = '<xsl:value-ofselect='HM_SITE_CODE' />'
  AND sp.hm_pcode = '<xsl:value-ofselect='HM_PCODE' />'
</xsl:if>
</xsl:template>
```

However, instead of `position()=1`, we need to substitute a parameter, such as `$1`.

How can we do this?

Answer

If you have a top-level parameter declared in your stylesheet, such as:

```
<xsl:stylesheet ... >
  <!-- declare top-level $foo parameter, default value to 5 -->
  <xsl:param name="foo" select="5"/>
  <xsl:template match="/">
  <xsl:if test="$foo=10">
    :
```

Then you can use the following methods on

`oracle.xml.parser.v2.XSLStyleSheet` to control parameters:

- `resetParams()`

- `setParam()`

To set the parameter named `foo` to the number 10, use the following:

```
myStylesheet.setParam("foo", "10");
```

To set `foo` to the string `ten`, you need to quote it:

```
myStylesheet.setParam("foo", "'ten'");
```

Question 2

If I need to pass parameters to the stylesheet in a Java program, what Java class must I use?

Currently, we use:

```
processXSL(XSLStylesheet xsl, XMLDocument xml)
```

What method can I use to pass the parameters?

Answer 2

See:

- `XSLStylesheet.setParam()`
- `XSLStylesheet.resetParams()`

How Can I Resolve the Error XSL-1009 Attribute 'XSL Version' Not Found in HTML?

We used [Note:104675.1](http://metalink.oracle.com) from <http://metalink.oracle.com>, that explains how to use the XDK to retrieve XML data from Oracle and transform it to HTML.

We can generate the XML output file but when we try to generate the HTML output by using the file, `Emp.xsl`, which has the following argument:

```
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

it shows error XSL-1009 ATTRIBUTE 'XSL VERSION' NOT FOUND IN 'HTML'

- What is the right argument for the XSL file? We tried `<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0" version="1.0">` and it works but the HTML output does not have any HTML tag at all, just pure data.
- I have never seen the HTML output generated from the XMLParser so I do not know whether it will generate HTML tags for me automatically or not.

What should the HTML output file look like?

Answer

You must add `xsl:version="1.0"` attribute to your `<html>` element.

What XPath Expression Will Retrieve Only Terminal Child Elements?

Can you tell me what XPath expression I should use to retrieve only terminal child elements (that is, elements which don't have any child elements) from a specified element. For example, I want to use an XPath expression to return only the `TABLE` child elements highlighted in red below:

```
<TABLE>
  <ID>1</ID>
  <NAME>1</NAME>
  <SIZE>1</SIZE>
  <COLUMNS>
    <COLUMN>
      <ID>1</ID>
      <NAME>Customers</NAME>
    </COLUMN>
    <COLUMN>
      <ID>c</ID>
      <NAME>Categories</NAME>
    </COLUMN>
  </COLUMNS>
  <DATE_CREATED>01/10/2000</DATE_CREATED>
</TABLE>
```

Answer 1

A possible solution is the following:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="TABLE">
  <xsl:apply-templates select="child::*[not(child:*)]"/>
</xsl:template>
</xsl:stylesheet>
```

Answer 2

The expression you want is:

```
/TABLE/*[count(child:**) = 0]
```

or

```
/TABLE/*[not (child:**)]
```

You can omit the child axis, so above expression is the same as:

```
/TABLE/*[count(*) = 0]
```

or

```
/TABLE/*[not (*)]
```

Child Attributes are Not Returned After Applying XSL Stylesheet

We are merging an XML document with its XSL stylesheet. Child attributes are not being returned when they are using syntax of type:

```
<xsl:value-of select="Foo/Bar"/>
```

in the XSL document. This seems to work fine in other XML parsers including XML Spy and Stylus.

Answer

The XPath expression `Foo/Bar` is only designed to select the value of the `<Bar>` element contained in the `<Foo>` element. It will return the concatenation of all text nodes in the nested content of that `<Bar>` element, but certainly is not designed to select any text values in attributes. For this, you'd need the syntax:

`Foo/Bar/@SomeAttr` to select one attribute and...

`Foo/Bar/@*` to select all the attributes of `<Bar>`.

Part II

Storing and Retrieving XML From the Database

Part II of this manual focuses on storing XML data in, and retrieving XML data from Oracle Database, how to use XML SQL Utility (XSU), SYS.XMLType, and Database URI-Reference to do these tasks. This section also describes how to use Oracle Text (*interMedia* Text) to fine tune and turbocharge your search and retrieval of XML data.

Part II contains the following chapters:

- [Chapter 5, "Database Support for XML"](#)
- [Chapter 6, "Database Uri-references"](#)
- [Chapter 7, "XML SQL Utility \(XSU\)"](#)
- [Chapter 8, "Searching XML Data with Oracle Text"](#)

Database Support for XML

This chapter contains the following sections:

- [What are the Oracle Native XML Database Features?](#)
- [XMLType Datatype](#)
 - [When to use XMLType](#)
 - [XMLType Storage in the Database](#)
 - [XMLType Functions](#)
 - [Manipulating XML Data in XMLType Columns](#)
 - [Selecting and Querying XML Data](#)
- [Indexing XMLType columns](#)
- [Java Access to XMLType \(oracle.xdb.XMLType\)](#)
- [DBMS_XMLGEN](#)
- [SYS_XMLGEN](#)
- [SYS_XMLAGG](#)
- [TABLE Functions](#)
- [Frequently Asked Questions \(FAQs\): XMLType](#)

What are the Oracle Native XML Database Features?

Oracle supports `XMLType`, a new system defined object type. `XMLType` has built-in member functions that offer a powerful mechanism to create, extract and index XML data. Users can also generate XML documents as `XMLType` instances dynamically using the SQL functions, `SYS_XMLGEN` and `SYS_XMLAGG`, and the PL/SQL package `DBMS_XMLGEN`.

[Table 5-1](#) summarizes the new XML features natively supported in Oracle.

Table 5-1 Oracle Native XML Support Feature Summary

XML Feature	Description
<code>XMLType</code>	<p>(new) <code>XMLType</code> is a system defined datatype with predefined member functions to access XML data. You can perform the following tasks with <code>XMLType</code>:</p> <ul style="list-style-type: none">■ Create columns of <code>XMLType</code> and use <code>XMLType</code> member functions on instances of the type. See "XMLType Datatype" on page 5-3.■ Create PL/SQL functions and procedures, with <code>XMLType</code> as argument and return parameters. See, "When to use XMLType" on page 5-8.■ Store, index, and manipulate XML data in <code>XMLType</code> columns. <p>Refer to "XMLType Datatype" on page 5-3.</p>
<code>DBMS_XMLGEN</code>	<p>(new) <code>DBMS_XMLGEN</code> is a PL/SQL package that converts the results of SQL queries to canonical XML format, returning it as <code>XMLType</code> or CLOB. <code>DBMS_XMLGEN</code> is implemented in C, and compiled in the database kernel. <code>DBMS_XMLGEN</code> is similar in functionality to <code>DBMS_XMLQuery</code> package.</p> <p>Refer to "DBMS_XMLGEN" on page 5-30.</p>

Table 5–1 Oracle Native XML Support Feature Summary (Cont.)

XML Feature	Description
SYS_XMLGEN	<p>(new) SYS_XMLGEN is a SQL function, which generates XML within SQL queries. DBMS_XMLGEN and other packages operate at a query level, giving aggregated results for the <i>entire</i> query. SYS_XMLGEN operates on a <i>single</i> argument inside a SQL query and converts the result to XML.</p> <p>SYS_XMLGEN takes in a scalar value, object type, or a XMLType instance to be converted to an XML document. It also takes an optional XMLGenFormatType object to specify formatting options for the result. SYS_XMLGEN returns a XMLType.</p> <p>Refer to "SYS_XMLGEN" on page 5-62.</p>
SYS_XMLAGG	<p>(new) SYS_XMLAGG is an aggregate function, which aggregates over a set of XMLType's. SYS_XMLAGG aggregates all the input XML documents/fragments and produces a single XML document by concatenating XML fragments, and adding a top-level tag.</p> <p>Refer to "SYS_XMLAGG" on page 5-71.</p>
UriTypes	<p>(new) The UriType family of types can store and query Uri-refs in the database. SYS.UriType is an abstract object type which provides functions to access the data pointed to by the URL. SYS.HttpUriType and SYS.DBUriType are subtypes of UriType. The HttpUriType can store HTTP URLs and the DBUriType can store intra-database references. You can also <i>define your own subtypes</i> of SYS.UriType to handle different URL protocols.</p> <p>UriFactory package: This is a factory package that can generate instances of these UriTypes automatically by scanning the prefix, such as, http:// or ftp:// etc. Users can also register their own subtype with UriFactory, specifying the supported prefix. For example, a subtype to handle the gopher protocol can be registered with UriFactory, specifying that URLs with the prefix "gopher://" are to be handled by your subtype. UriFactory now generates the registered subtype instance for any URL starting with that prefix.</p> <p>See Chapter 6, "Database Uri-references".</p>

XMLType Datatype

XMLType is a new server datatype that can be used as columns in tables, and views. Variables of XMLType can be used in PL/SQL stored procedures as parameters, return values, and so on. You can use XMLType *inside the server*, in PL/SQL, SQL and Java. It is not currently supported through OCI.

XMLType includes useful built-in member functions that operate on XML content. For example, function *Extract* extracts a specific node(s) from an XMLType instance.

New SQL functions such as SYS_XMLGEN that return XML documents return them as XMLType. This provides strong typing in SQL statements. You can use XMLType in SQL queries in the same way as any other user-defined datatypes in the system.

Note: In this release, XMLType is only supported in the server in SQL, PL/SQL, and Java. To use XMLType on the client side, use Oracle Call Interface (OCI) or Oracle C++ Call Interface (OCCI), and such functions as `getClobVal()` or other functions on it to retrieve the complete XML document.

How to use XMLType

XMLType can be used to create table columns. The `createXML()` static function in the XMLType can be used to create XMLType instances for insertion. By storing your XML documents as XMLType, XML content can be readily searched using standard SQL queries.

We will show some simple examples on how to create an XMLType column and use it in a SQL statement.

Example of Creating XMLType columns

The XMLType column can be created like any other user-defined type column,

```
CREATE TABLE warehouses(  
  warehouse_id NUMBER(3),  
  warehouse_spec SYS.XMLTYPE,  
  warehouse_name VARCHAR2(35),  
  location_id NUMBER(4));
```

Example of Inserting values into an XMLType column

To insert values into the XMLType column, you would need to bind an XMLType instance. An XMLType instance can be easily created from a varchar or a CLOB by using the `createXML()` static function of the XMLType.

```
INSERT INTO warehouses (warehouse_id, warehouse_spec)  
VALUES (1001, sys.XMLType.createXML(  
  '<Warehouse whNo="100">  
    <Building>Owned</Building>  
  </Warehouse>');
```

In this example, we are creating an XMLType instance from a string literal. The input to the `createXML` could be any expression which returns a `varchar2` or a `CLOB`.

The `createXML` function also checks to make sure that the input XML is well-formed. It does not check for validity of the XML.

Example of Using XMLType in a SQL Statement

The following simple SELECT statement shows how you can use XMLType in a SQL statement:-

```
SELECT
  w.warehouse_spec.extract('/Warehouse/Building/text()').getStringVal()
  "Building"
FROM warehouses w
```

where `warehouse_spec` is a XMLType column operated on by member function `Extract()`. The result of this simple query is a string (varchar2):

```
Building
-----
Owned
```

See Also: ["How to use XMLType"](#) on page 5-4.

Example of Updating an XMLType column

In this release, an XML document in an XMLType is stored packed in a CLOB. Consequently updates, have to replace the document in place. We do not support piece-wise update of the XML for this release.

To update an XML document, you would fire a standard SQL update statement, except that you would bind an XMLType instance.

```
UPDATE warehouses SET warehouse_spec =
  sys.XMLType.createXML(
    '<Warehouse whono="200">
      <Building>Leased</Building>
    </Warehouse>');
```

In this example, we are creating an XMLType instance from a string literal and updating the `warehouse_spec` column with the new value. Note that any triggers would get fired on the update statement and you can see and modify the XML value inside the triggers.

Example of Deleting a row containing an XMLType column

Deleting a row containing an XMLType column is no different from any other datatype.

You can use the `Extract` and `ExistsNode` functions to identify rows to delete as well. For example to delete all warehouse rows for which the warehouse building is Leased, we can write a statement such as,

```
DELETE FROM warehouses e
WHERE e.warehouse_spec.extract('//Building/text()').getStringVal()
= 'Leased';
```

Guidelines for using XMLType Columns

The following are guidelines for storing XML data in `XMLType` columns:

- **Define column XMLType.** First, define a column of `XMLType`. You can include optional storage characteristics with the column definition.
- **Create an XMLType instance.** Use the `XMLType` constructor to create the `XMLType` instance before inserting into the column. You can also use the `SYS_XMLGEN` and `SYS_XMLAGG` functions to directly create instances of `XMLType`. See "[SYS_XMLGEN Example 3: Converting XMLType Instance](#)" on page 5-67 and "[SYS_XMLAGG Example 2: Aggregating XMLType Instances Stored in Tables](#)" on page 5-73.
- **Select or extract a particular XMLType instance.** You can select out the `XMLType` instance from the column. `XMLType` offers a choice of member functions, such as, `extract()` and `existsNode()`, to extract a particular node or check to see if a node exists, respectively. See [Table 5-3, "XMLType Member and Static Functions"](#).

See Also:

- "[XMLType Query Example 6 — Extract fragments from XMLType](#)" on page 28
- "[XMLType Query Example 1 — Retrieve an XML Document as a CLOB](#)" on page 5-20
- **You can define an Oracle Text index.** You can define an Oracle Text (*interMedia Text*) index on `XMLType` columns. This enables you to use `CONTAINS`, `HASPATH`, `INPATH`, and other text operators on the column. All the text operators and index functions that operate on LOB columns, also work on `XMLType` columns.

See Also:

- ["Indexing XMLType columns"](#) on page 30
- [Chapter 8, "Searching XML Data with Oracle Text"](#)
- *Oracle9i Application Developer's Guide - Large Objects (LOBs)*

Benefits of XMLType

Using XMLType has the following advantages:

- It brings the worlds of XML and SQL together as it enables:
 - SQL operations on XML content
 - XML operations on SQL content
- It is convenient to implement as it includes built-in functions, indexing support, navigation, and so on.

XMLType Interacts with Other SQL Constructs

You can use XMLType in SQL statements combined with other columns and datatypes. For example, you can query XMLType columns, join the result of the extraction with a relational column, and then Oracle can determine an *optimal* way to execute these queries.

You Can Select a Tree or Serialized Format for Your Resulting XML

XMLType is optimized to not materialize the XML data into a tree structure unless needed. Hence when SQL selects XMLType instances inside queries, only a serialized form is exchanged across function boundaries. These are exploded into tree format only when operations such as `extract()` and `existsNode()` are performed. The internal structure of XMLType is also an optimized DOM-like tree structure.

You Can Create functional indexes and Text indexes on XMLType

Oracle Text index has been enhanced to support XMLType columns as well. You can create functional indexes on Existsnode and Extract functions as well to speed up query evaluation.

See Also: [Chapter 8, "Searching XML Data with Oracle Text"](#)

When to use XMLType

Use XMLType in the following cases:

- You need to store XML as a whole in the database and retrieve it.
- You need SQL queriability on some or the whole document. The functions ExistsNode and Extract provide the necessary SQL queriability over XML documents.
- You need strong typing inside SQL statements and PL/SQL functions. Strong typing implies that you ensure that the values passed in are XML values and not any arbitrary text string.
- You need the XPath functionality provided by Extract and ExistsNode functions to work on your XML document. Note that the XMLType uses the built-in C XML parser and processor and hence would provide better performance and scalability when used inside the server.
- You need indexing on XPath searches on documents. XMLType provides member functions that can be used to create functional indexes to optimize searches.
- You do not need piecewise updates of the document.
- Shield applications from storage models - In future releases, the XMLType would support different storage alternatives. Using XMLType instead of CLOBs or relational storage, allows applications to gracefully move to various storage alternatives later, without affecting any of the query or DML statements in the application.
- Preparing for future optimizations - All new functionality related to XML will only support the XMLType. Since the server is natively aware that the XMLType can only store XML data, better optimizations and indexing techniques can be done in the future. By writing applications to use XMLType, these optimizations and enhancements can be easily achieved in the future without rewriting the application.

XMLType Storage in the Database

In this release, XMLType offers a single CLOB storage option. In future releases, Oracle may provide other storage options, such as BLOBs, NCLOBs, and so on.

When you create an XMLType column, a hidden CLOB column is automatically created to store the XML data. The XMLType column itself becomes a virtual column over this hidden CLOB column. It is not possible to directly access the

CLOB column, however, you can set the storage characteristics for the column using the XMLType storage clause.

You cannot create VARRAYs of XMLType and store it in the database since VARRAYs do not support CLOBs when stored in tables.

Note: You cannot create columns of VARRAY types which contain XMLType. This is because Oracle does not support LOB locators inside VARRAYs, and XMLType (currently) always stores the XML data in a CLOB.

XMLType Creation Example 1 — Creating XMLType Columns

As explained earlier, you can create XMLType columns by simply using the XMLType as the datatype.

The following statement creates a purchase order document column of XMLType.

```
CREATE TABLE po_xml_tab(  
    poId number,  
    poDoc SYS.XMLTYPE);
```

XMLType Creation Example 2 — Adding XMLType Columns

You can alter tables to add XMLType columns as well. This is similar to any other datatype,

The following statement adds a new customer document column to the table,

```
ALTER TABLE po_xml_tab add (custDoc sys.XMLTYPE);
```

XMLType Creation Example 3 — Dropping XMLType Columns

You can alter tables to drop XMLType columns, similar to any other datatype,

The following statement drops the custDoc column.

```
ALTER TABLE po_xml_tab drop (custDoc sys.XMLTYPE);
```

Specifying Storage Characteristics on XMLType Columns

As previously mentioned, the XML data in a XMLType column is stored as a CLOB column. You can also specify LOB storage characteristics for the CLOB column. In the previous example, the warehouse spec column is an XMLType column.

Figure 5–1 illustrates the XMLType storage clause syntax.

Figure 5–1 XMLType Storage Clause Syntax

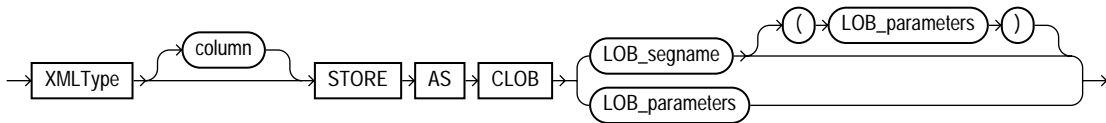


Table 5–2 explains the XMLType storage clause syntax.

Table 5–2 XMLType Storage Clause Syntax Description - See Figure 5–1

Syntax Member	Description
<i>column</i>	Specifies the LOB column name or LOB object attribute for which you are explicitly defining tablespace and storage characteristics that are different from those of the table. Oracle automatically creates a system-managed index for each <i>column</i> you create.
<i>LOB_segname</i>	Specify the name of the LOB data segment. You cannot use <i>LOB_segname</i> if you specify more than one <i>LOB_item</i> .
<i>LOB_parameters</i>	<p>The <i>LOB_parameters</i> clause lets you specify various elements of LOB storage.</p> <ul style="list-style-type: none"> ▪ ENABLE STORAGE IN ROW: If you enable storage in row, the LOB value is stored in the row (inline) if its length is less than approximately 4000 bytes minus system control information. This is the default. Restriction: For an index-organized table, you cannot specify this parameter unless you have specified an OVERFLOW segment in the <i>index_org_table_clause</i>. ▪ DISABLE STORAGE IN ROW: If you disable storage in row, the LOB value is stored out of line (outside of the row) regardless of the length of the LOB value. Note: The LOB locator is always stored inline (inside the row) regardless of where the LOB value is stored. You cannot change the value of STORAGE IN ROW once it is set except by moving the table. See <i>Oracle9i SQL Reference</i>, ALTER TABLE — move_table_clause. ▪ CHUNK integer: Specifies bytes to be allocated for LOB manipulation. If <i>integer</i> is not a multiple of the database block size, Oracle rounds up (in bytes) to the next multiple. If database block size is 2048 and <i>integer</i> is 2050, Oracle allocates 4096 bytes (2 blocks). Maximum value is 32768 (32K). Default CHUNK size is one Oracle database block. You cannot change the value of CHUNK once it is set. Note: The value of CHUNK must be less than or equal to the value of NEXT (either the default value or that specified in the <i>storage_clause</i>). If CHUNK exceeds the value of NEXT, Oracle returns an error. ▪ PCTVERSION integer: Specify the maximum percentage of overall LOB storage space used for creating new versions of the LOB. The default value is 10, meaning that older versions of the LOB data are not overwritten until 10% of the overall LOB storage space is used.

You can specify storage characteristics on this column when creating the table as follows:

```
CREATE TABLE po_xml_tab(
  poid NUMBER(10),
  poDoc SYS.XMLTYPE
)
XMLType COLUMN poDocument
```

```
STORE AS CLOB (  
    TABLESPACE lob_seg_ts  
    STORAGE (INITIAL 4096 NEXT 4096)  
    CHUNK 4096 NOCACHE LOGGING  
);
```

The storage clause is also supported while adding columns to the table. If you want to add a new XMLType column to this table and specify the storage clause for that you can do the following:-

```
ALTER TABLE po_xml_tab add(  
    custDoc SYS.XMLTYPE  
)  
XMLType COLUMN custDoc  
STORE AS CLOB (  
    TABLESPACE lob_seg_ts  
    STORAGE (INITIAL 4096 NEXT 4096)  
    CHUNK 4096 NOCACHE LOGGING  
);
```

See also: *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for more information about LOB storage options.

Specifying Constraints on XMLType Columns

You can specify the NOT NULL constraint on a XMLType column. For example:

```
CREATE TABLE po_xml_tab (  
    poId number(10),  
    poDoc sys.XMLType NOT NULL  
);
```

prevents inserts such as:

```
INSERT INTO po_xml_tab (poDoc) VALUES (null);
```

You can also use the ALTER TABLE statement to change the NOT NULL information of a XMLType column, in the same way you would for other column types:

```
ALTER TABLE po_tab MODIFY (poDoc NULL);  
ALTER TABLE po_tab MODIFY (poDoc NOT NULL);
```

Default values and other check constraints are not supported on this datatype.

XMLType Functions

Oracle has introduced two new SQL functions `ExistsNode` and `Extract` that operator on XMLType values.

The `existsNode()` function uses the `XMLType.existsNode()` member function for its implementation. The syntax of the `ExistsNode` function is:

```
existsNode ( "XMLType_instance" IN sys.XMLType,  
            "XPath_string" IN VARCHAR2) RETURN NUMBER
```

`extract()` function applies an XPath expression and returns an XMLType containing the resultant XML fragment. The syntax is:

```
extract ( "XMLType_instance" IN sys.XMLType,  
         "XPath_string" IN VARCHAR2) RETURN sys.XMLType;
```

Note: In this release, `existsNode()` and `extract()` SQL functions only use the functional implementation. In future releases, these functions will use new indices and be further optimized.

[Table 5–3](#) lists all the XMLType SQL and member functions, their syntax and descriptions.

You can use the SQL functions instead of the member functions `ExistsNode` and `Extract` inside any SQL statement. All the XMLType functions use the built-in C parser and processor to parse the XML data, validate it and apply XPath expressions over it. It also uses an optimized in-memory DOM tree to do processing (such as `Extract`).

Table 5–3 XMLType Member and Static Functions

XMLType Function	Syntax Summary	Description
<code>createXML()</code>	STATIC FUNCTION <code>createXML(xmlval IN varchar2)</code> RETURN <code>sys.XMLType</code> deterministic	Static function to create the <code>XMLType</code> instance from a string. Checks for well-formed XML value. PARAMETERS: <code>xmlval</code> (IN) - string containing the XML document. RETURNS: A <code>XMLType</code> instance. String must contain a well-formed XML document. See also " XMLType Query Example 6 — Extract fragments from XMLType " on page 5-28, and other examples in this chapter.
<code>createXML()</code>	STATIC FUNCTION <code>createXML(xmlval IN clob)</code> RETURN <code>sys.XMLType</code> deterministic	Static function to create the <code>XMLType</code> instance from a CLOB. Checks for well-formed XML value. PARAMETERS: <code>xmlval</code> (IN) - CLOB containing the XML document RETURNS: A <code>XMLType</code> instance. CLOB must contain a well-formed XML document. See " XMLType Query Example 2 — Using extract() and existsNode() " on page 5-24 and other examples in this chapter.
<code>existsNode()</code>	MEMBER FUNCTION <code>existsNode(xpath IN varchar2)</code> RETURN number deterministic	Given an XPath expression, checks if the XPath applied over the document can return any valid nodes. PARAMETERS: <code>xpath</code> (IN) - the XPath expression to test RETURNS: 0 if the XPath expression does not return any nodes else 1. If the XPath string is null or the document is empty, then a value of 0 is returned. See also " XMLType Query Example 2 — Using extract() and existsNode() " on page 5-24.
<code>extract()</code>	MEMBER FUNCTION <code>extract(xpath IN varchar2)</code> RETURN <code>sys.XMLType</code> deterministic	Given an XPath expression, applies the XPath to the document and returns the fragment as a <code>XMLType</code> PARAMETERS: <code>xpath</code> (IN) - the XPath expression to apply RETURNS: A <code>XMLType</code> instance containing the result node(s). If the XPath does not result in any nodes, then the result is NULL. See also " XMLType Query Example 6 — Extract fragments from XMLType " on page 5-28.

Table 5–3 XMLType Member and Static Functions(Cont.)

XMLType Function	Syntax Summary	Description
<code>isFragment()</code>	MEMBER FUNCTION <code>isFragment()</code> RETURN number	Checks if the document is really a fragment. A fragment might be present, if an <code>EXTRACT</code> or other operation was done on an XML document that may have resulted in many nodes. RETURNS: A numerical value 1 or 0 indicating if the <code>XMLType</code> instance contains a fragment or a well-formed document. See also " XMLType Query Example 6 — Extract fragments from XMLType " on page 5-28.
<code>getClobVal()</code>	MEMBER FUNCTION <code>getClobVal()</code> RETURN clob deterministic	Gets the document as a CLOB. RETURNS: A CLOB containing the serialized XML representation. Free the temporary CLOB after use. See also: " XMLType Query Example 1 — Retrieve an XML Document as a CLOB " on page 5-54.
<code>getStringVal()</code>	MEMBER FUNCTION <code>getStringVal()</code> RETURN varchar2 deterministic	Gets the XML value as a string. RETURNS: A string containing the serialized XML representation, or in case of text nodes, the text itself. If the XML document is bigger than the maximum size of <code>VARCHAR2</code> , (4000 bytes), an error is raised at run time. See " XMLType Delete Example 1 — Deleting Rows Using extract " and also, " How to use XMLType " on page 5-4.
<code>getNumberVal()</code>	MEMBER FUNCTION <code>getNumberVal()</code> RETURN number deterministic	Gets the numeric value pointed to by the <code>XMLType</code> as a number RETURNS: A number formatted from the text value pointed to by the <code>XMLType</code> instance. The <code>XMLType</code> must point to a valid text node that contains a numeric value. See also: " XMLType Query Example 2 — Using extract() and existsNode() " on page 5-24.

See Also: "[How to use XMLType](#)" examples starting on page 5-4, for ideas on how you can use `extract()`, `existsNode()`, `getClobVal()`, and other functions.

Manipulating XML Data in XMLType Columns

Since `XMLType` is a user-defined data type with functions defined on it, you can invoke functions on `XMLType` and obtain results. You can use `XMLType` wherever you use a user-defined type. This includes columns of tables, views, trigger body, type definitions, and so on.

You can perform the following manipulations (DML) on XML data in `XMLType` columns:

- Insert XML data
- Update XML data
- Delete XML data

Inserting XML Data into XMLType Columns

You can insert data into `XMLType` columns in the following ways:

- By using the `INSERT` statement (in SQL, PL/SQL, C(OCI), and Java)
- By using `SQL*Loader`

The `XMLType` columns can only store well-formed XML documents. Fragments and other non-well formed XML cannot be stored in such columns.

Using INSERT Statements

If you use the `INSERT` statement to insert XML data into `XMLType`, you need to first create XML documents to perform the insert with. You can create the insertable XML documents as follows:

1. Using `XMLType` constructors, `SYS.XMLType.createXML()`. This can be done in SQL, PL/SQL, C(OCI), and Java.
2. Using `SYS_XMLGEN` and `SYS_XMLAGG` SQL functions. This can be done in PL/SQL, SQL, C(OCI), and Java.

XMLType Insert Example 1- Using createXML() with CLOB

The following examples use `INSERT...SELECT` and the `XMLType.createXML()` construct to first create an XML document and then insert the document into `XMLType` columns.

For example, if the `po_clob_tab` is a table containing a CLOB that stores an XML document,

```

CREATE TABLE po_clob_tab
(
  poid number,
  poClob CLOB
);

-- some value is present in the po_clob_tab
INSERT INTO po_clob_tab
VALUES(100, '<?xml version="1.0"?>
          <PO pono="1">
            <PNAME>Po_1</PNAME>
            <CUSTNAME>John</CUSTNAME>
            <SHIPADDR>
              <STREET>1033, Main Street</STREET>
              <CITY>Sunnyvalue</CITY>
              <STATE>CA</STATE>
            </SHIPADDR>
          </PO>');

```

Now you can insert a purchase order XML document into table, `po_tab` by simply creating an XML instance from the CLOB data stored in the other `po_clob_tab`,

```

INSERT INTO po_xml_tab
SELECT poid, sys.XMLType.createXML(poClob)
FROM po_clob_tab;

```

Note that we could have gotten the clob value from any expression including functions which can create temporary CLOBs or select out CLOBs from other table or views.

XMLType Insert Example 2 - Using `createXML()` with string

This example inserts a purchase order into table, `po_tab` using the `createXML()` construct.

```

insert into po_xml_tab
VALUES(100, sys.XMLType.createXML('<?xml version="1.0"?>
          <PO pono="1">
            <PNAME>Po_1</PNAME>
            <CUSTNAME>John</CUSTNAME>
            <SHIPADDR>
              <STREET>1033, Main Street</STREET>
              <CITY>Sunnyvalue</CITY>
              <STATE>CA</STATE>
            </SHIPADDR>
          </PO>');

```

```
</PO>' ));
```

Here the XMLType was created using the createXML function by passing in a string literal.

XMLType Insert Example 3 - Using SYS_XMLGEN()

This example inserts PurchaseOrder into table, po_tab by generating it using the SYS_XMLGEN() SQL function which is explained later in this chapter. Assume that the PO is an object view that contains a purchase order object. The whole definition of the PO view is given in ["DBMS_XMLGEN Example 5: Generating a Purchase Order From the Database in XML Format"](#).

```
INSERT into po_xml_tab
  SELECT SYS_XMLGEN(value(p),
                    sys.xmlgenformatType.createFormat('PO'))
  FROM po p
  WHERE p.pono=2001;
```

The SYS_XMLGEN creates an XMLType from the purchase order object which is then inserted into the po_xml_tab table.

Updating XML Data in XMLType Columns

You can only update the whole XML document. You can perform the update in SQL, PL/SQL, C(OCI) or Java.

See also ["XMLType Java Example 4: Updating an Element in XMLType Column"](#) on page 5-35, for updating XMLType through Java.

XMLType Update Example 1 — Updating Using createXML()

This example updates the XMLType using the createXML() construct. It updates only those documents whose purchase order number is 2001.

```
update po_xml_tab e
set e.poDoc = sys.XMLType.createXML(
'<?xml version="1.0"?>
<PO pono="2">
  <PNAME>Po_2</PNAME>
  <CUSTNAME>Nance</CUSTNAME>
  <SHIPADDR>
    <STREET>2 Avocet Drive</STREET>
    <CITY>Redwood Shores</CITY>
    <STATE>CA</STATE>
```

```

    </SHIPADDR>
  </PO>')
WHERE e.po.extract('/PO/PONO/text()').getNumberVal() = 2001;

```

Note: UPDATES, currently are supported only at the document level. So to update a piece of a particular document, you would have to update the entire document.

Deleting XML Data

DELETEs on the row containing the XMLType column are handled in the same way as any other datatype.

XMLType Delete Example 1 — Deleting Rows Using extract

For example, to delete all purchase order rows with a purchase order name of “Po_2”, you can execute a statement such as:

```

DELETE from po_xml_tab e
WHERE e.poDoc.extract('/PO/PNAME/text()').getStringVal()='Po_2';

```

Using XMLType Inside Triggers

You can use the NEW and OLD binds inside triggers to read and modify the XMLType column values. In the case of INSERTs and UPDATE statements, you can modify the NEW value to change the value being inserted.

XMLType Trigger Example 1 -

For instance, you can write a trigger to change the purchase order to be a different one if it does not contain a shipping address.

```

CREATE OR REPLACE TRIGGER po_trigger
  BEFORE insert or update on po_xml_tab for each row
  pono Number;
begin

  if INSERTING then
    if :NEW.poDoc.existsnode('//SHIPADDR') = 0 then
      :NEW.poDoc := sys.xmltype.createxml('<PO>INVALID_PO</PO>'); end if;
    end if;

    -- when updating, if the old poDoc has purchase order number

```

```

-- different from the new one then make it an invalid PO.
if UPDATING then

    if :OLD.poDoc.extract('//PONO/text()').getNumberVal() !=
       :NEW.poDoc.extract('//PONO/text()').getNumberVal() then

        :NEW.poDoc := sys.xmltype.createXML('<PO>INVALID_PO</PO>');
    end if;
end if;
end;
/

```

This example is of course, only for illustration purposes. You can use the XMLType value to perform useful operations inside the trigger, such as validation of business logic or rules that the XML document should conform to, auditing..

Selecting and Querying XML Data

You can query XML Data from XMLType columns in the following ways:

- By selecting XMLType columns through SQL, PL/SQL, C(OCI), or Java.
- By querying XMLType columns directly and using `extract()` and/or `existsNode()`.
- By using Text operators to query the XML content

Selecting XML data

You can select the XMLType data through PL/SQL or Java. You can also use the `getClobVal()`, `getStringVal()` or `getNumberVal()` functions to get out the XML as a CLOB, varchar or a number respectively.

XMLType Query Example 1 — Retrieve an XML Document as a CLOB

This example shows how to select an XMLType column through SQL*Plus

```

set long 2000

select e.poDoc.getClobval() AS poXML
from po_xml_tab e;

POXML
-----
<?xml version="1.0"?>

```

```

<PO pono="2">
  <PNAME>Po_2</PNAME>
  <CUSTNAME>Nance</CUSTNAME>
  <SHIPADDR>
    <STREET>2 Avocet Drive</STREET>
    <CITY>Redwood Shores</CITY>
    <STATE>CA</STATE>
  </SHIPADDR>
</PO>

```

Querying XML data

We can query the XMLType data and extract portions of it using the ExistsNode and Extract functions. Both these functions use a limited set of the W3C standard XPath to navigate the document.

Using XPath Expressions for Searching

XPath is a W3C standard way to navigate XML documents. XPath models the XML document as a tree of nodes. It provides a rich set of operations to “walk” the tree and to apply predicates and node test functions. Applying an XPath expression to an XML document can result in a set of nodes. For instance, /PO/PONO selects out all the “PONO” child elements under the “PO” root element of the document.

Here are some of the common constructs used in XPath:-

The “/” denotes the root of the tree in an XPath expression. e.g. /PO refers to the child of the root node whose name is “PO”.

The “/” is also used as a path separator to identify the children node of any given node. e.g. /PO/PNAME identifies the purchase order name element which is a child of the root element.

The “//” is used to identify all descendants of the current node. e.g. PO//ZIP matches any zip code element under the “PO” element.

The “*” is used as a wildcard to match any child node. e.g. /PO/*/STREET would match any street element that is a grandchild of the “PO” element.

The [] are used to denote predicate expressions. XPath supports a rich list of binary operators such as OR, AND and NOT. e.g. /PO[PONO=20 and PNAME="PO_2"]/SHIPADDR selects out the shipping address element of all purchase orders, whose purchase order number is 20 and whose purchase order name is “PO_2”

The [] is also used for denoting an index into a list. For instance, /PO/PONO[2] identifies the second purchase order number element under the "PO" root element.

Supported XPath constructs

Oracle's Extract and ExistsNode functions support a limited set of XPath expressions. XPath constructs supported in this release are:

- Child traversals /PO/SHIPADDR/STREET
- Attribute traversals /PO/@PONO
- Index access /PO/PONO[2]
- Wild card searches /PO/*/STREET
- Descendant searches PO//STREET
- Node test functions - /PO/PNAME/text()

Only the non-order dependant axes such as the child, descendant axes are supported. No sibling or parent axes are supported.

Note: Extract and ExistsNode functions do not yet support multi-byte character sets.

Predicates are Not Supported For this release, XMLType does not support any predicates. If you need predicate support, you can rewrite the function into multiple functions with the predicates expressed in SQL, when possible.

Finally, the XPath must identify a single or a set of element, text or attribute nodes. The result of the XPath cannot be a boolean expression.

existsNode() Function with XPath

The ExistsNode function on XMLType, checks if the given XPath evaluation results in at least a single XML element or text node. If so, it returns the numeric value 1 otherwise it returns a 0. For example, consider an XML document such as:

```
<PO>
  <PONO>100</PONO>
  <PNAME>Po_1</PNAME>
  <CUSTOMER CUSTNAME="John" />
  <SHIPADDR>
    <STREET>1033, Main Street</STREET>
    <CITY>Sunnyvalue</CITY>
    <STATE>CA</STATE>
  </SHIPADDR>
</PO>
```


An XPath expression such as `/PO/PNAME`, results in a single node and hence the `ExistsNode` will return true for that XPath. This is the same with `/PO/PNAME/text()` which results in a single text node. The XPath, `/PO/@pono` also returns a value.

An XPath expression such as, `/PO/POTYPE` does not return any nodes and hence an `ExistsNode` on this would return the value 0.

Hence, the `ExistsNode()` member function can be directly used in the following ways:

- In queries as will be shown in the next few examples
- To create functional indexes to speed up evaluation of queries

Extract Function with XPath

The `Extract` function on `XMLType`, extracts the node or a set of nodes from the document identified by the XPath expression. The extracted nodes may be elements, attributes or text nodes. When extracted out all text nodes are collapsed into a single text node value.

The `XMLType` resulting from applying an XPath through `Extract` need not be a well-formed XML document but can contain a set of nodes or simple scalar data in some cases. You can use the `getStringVal()` or `getNumberVal()` methods on `XMLType` to extract this scalar data.

For example, the XPath expression `/PO/PNAME` identifies the `PNAME` element inside the XML document shown above. The expression `/PO/PNAME/text()` on the other hand refers to the text node of the `PNAME` element. Note that the latter is still considered an `XMLType`. i.e. `EXTRACT(poDoc, '/PO/PNAME/text()')` still returns an `XMLType` instance though the instance may actually contain only text. You can use the `getStringVal()` to get the text value out as a `varchar2` result.

Use the `text()` node test function to identify text nodes in elements before using the `getStringVal()` or `getNumberVal()` to convert them to SQL data. Not having the `text()` node would produce an XML fragment. For example, the XPath expression `/PO/PNAME` identifies the fragment `<PNAME>PO_1</PNAME>` whereas the expression `/PO/PNAME/text()` identifies the text value "PO_1".

You can use the index mechanism to identify individual elements in case of repeated elements in an XML document. For example if we had an XML document such as,

```
<PO>
  <PONO>100</PONO>
```

```
<PONO>200</PONO>
</PO>
```

you can use `//PONO[1]` to identify the first “PONO” element (with value 100) and `//PONO[2]` to identify the second “PONO” element in the document.

The result of the extract is always an `XMLType`. If applying the XPath produces an empty set, then the Extract returns a `NULL` value.

Hence, the `extract()` member function can be used in a number of ways, including the following:

- Extracting numerical values on which functional indexes can be created to speed up processing
- Extracting collection expressions to be used in the FROM clause of SQL statements
- Extracting fragments to be later aggregated to produce different documents

XMLType Query Example 2 — Using `extract()` and `existsNode()`

Assume the `po_xml_tab` table which contains the purchase order id and the purchase order XML columns - and assume that the following values are inserted into the table,

```
INSERT INTO po_xml_tab values (100,
    sys.xmltype.createxml('<?xml version="1.0"?>
        <PO>
            <PONO>221</PONO>
            <PNAME>PO_2</PNAME>
        </PO>' ));

INSERT INTO po_xml_tab values (200,
    sys.xmltype.createxml('<?xml version="1.0"?>
        <PO>
            <PONAME>PO_1</PONAME>
        </PO>' ));
```

Now we can extract the numerical values for the purchase order numbers using the `EXTRACT` function.

```
SELECT e.poDoc.extract('//PONO/text()').getNumberVal() as pono
FROM po_xml_tab e
WHERE e.poDoc.existsnode('/PO/PONO') = 1 AND poId > 1;
```

Here `extract()` extracts the contents of tag, purchase order number, “PONO”. `existsNode()` finds those nodes where there exists “PONO” as a child of “PO”.

Note the use of the `text()` function to return only the text nodes. The `getNumberVal()` function can convert only text values into numerical quantity.

See Also: ["XMLType Functions"](#) on page 5-13.

XMLType Query Example 3 — Querying Transient XMLtype data

The following example shows how you can select out the XML data and query it inside PL/SQL.

```
-- create a transient instance from the purchase order table and then
-- perform some extraction on it...
declare
  poxml SYS.XMLType;
  cust SYS.XMLType;
  val VARCHAR2;
begin

  -- select the adt instance
  select poDoc into poxml
    from po_xml_tab p where p.poid = 100;

  -- do some traversals and print the output
  cust := poxml.extract('//SHIPADDR');

  -- do something with the customer XML fragment
  val := cust.getStringVal();
  dbms_output.put_line(' The customer XML value is '|| val);

end;
/
```

XMLType Query Example 4 — Extracting data from XML

The following example shows how you can extract out data from a purchase order XML and insert it into a SQL relation table.

Assume the following relational tables,

```
CREATE TABLE cust_tab
(
  custid number primary key,
  custname varchar2(20)
```

```
);

insert into cust_tab values (1001, "John Nike");

CREATE table po_rel_tab
(
  pono number,
  pname varchar2(100),
  custid number references cust_tab
  shipstreet varchar2(100),
  shipcity varchar2(30),
  shipzip varchar2(20)
);
```

You can write a simple PL/SQL block to transform any XML of the form,

```
<?xml version = '1.0'?>
<PO>
  <PONO>2001</PONO>
  <CUSTOMER CUSTNAME="John Nike"/>
  <SHIPADDR>
    <STREET>323 College Drive</STREET>
    <CITY>Edison</CITY>
    <STATE>NJ</STATE>
    <ZIP>08820</ZIP>
  </SHIPADDR>
</PO>
```

into the relational tables, using the Extract functions.

Here is a SQL example, (assuming that the XML described above is present in the po_xml_tab) -

```
insert into po_rel_tab as
select p.poDoc.extract('/PO/PONO/text()').getnumberval(),
       p.poDoc.extract('/PO/PNAME/text()').getstringval(),
       -- get the customer id corresponding to the customer name
       ( SELECT custid
         FROM   cust_tab c
         WHERE  c.custname =
              p.poDoc.extract('/PO/CUSTOMER/@CUSTNAME').getstringval()
       ),
       p.poDoc.extract('/PO/SHIPADDR/STREET/text()').getstringval(),
       p.poDoc.extract('//CITY/text()').getstringval(),
       p.poDoc.extract('//ZIP/text()').getstringval(),
```

```
from po_xml_tab p;
```

The po_tab would now have the following values,

PONO	PNAME	CUSTID	SHIPSTREET	SHIPCITY	SHIPZIP
2001		1001	323 College Drive	Edison	08820

Note how the PNAME is null, since the input XML document did not have the element called PNAME under PO. Also, note that we have used the //CITY to search for the city element at any depth.

We can do the same in an equivalent fashion inside a PL/SQL block-

```
declare
  poxml SYS.XMLType;
  cname varchar2(200);
  pono number;
  pname varchar2(100);
  shipstreet varchar2(100);
  shipcity varchar2(30);
  shipzip varchar2(20);

begin
  -- select the adt instance
  select poDoc into poxml from po_xml_tab p;

  cname := poxml.extract('//CUSTOMER/@CUSTNAME').getStringval();

  pono := poxml.extract('//PO/PONO/text()').getnumberval(),
  pname := poxml.extract('//PO/PNAME/text()').getStringval(),
  shipstreet := poxml.extract('//PO/SHIPADDR/STREET/text()').getStringval(),
  shipcity := poxml.extract('//CITY/text()').getStringval(),
  shipzip := poxml.extract('//ZIP/text()').getStringval(),

  insert into po_rel_tab
    values (pono, pname,
           (select custid from cust_tab c where custname = cname),
           shipstreet, shipcity, shipzip);

end;
/
```

XMLType Query Example 5 — Using `extract()` to Search

Using `Extract`, `ExistsNode` functions, you can perform a variety of operations on the column, as follows:

```
select e.poDoc.extract('/PO/PNAME/text()').getStringVal() PNAME
from po_xml_tab e
where e.poDoc.existsNode('/PO/SHIPADDR') = 1 and
      e.poDoc.extract('/PONO/text()').getNumberVal() = 300 and
      e.poDoc.extract('//@CUSTNAME').getStringVal() like '%John%';
```

This SQL statement extracts the purchase order name “PNAME” from the purchase order element PO, from all the documents which contain a shipping address and whose purchase order number is 300 and the customer name “CUSTNAME” contains the string “John”.

XMLType Query Example 6 — Extract fragments from XMLType

The `extract()` member function *extracts* the nodes identified by the XPath expression and returns a `XMLType` containing the fragment. Here, the result of the traversal may be a set of nodes, a singleton node, or a text value. You can check if the result is a fragment by using the `isFragment()` function on the `XMLType`. For example:

```
select e.po.extract('/PO/SHIPADDR/STATE').isFragment()
from foo_tab e;
```

Note: You cannot insert fragments into `XMLType` columns. You can use the `SYS_XMLGEN` function to convert a fragment into a well formed document by adding an enclosing tag. See "[SYS_XMLGEN](#)" on page 5-62. You can, however, query further on the fragment using the various `XMLType` functions.

The previous SQL statement would return 0, since the extraction `/PO/SHIPADDR/STATE` returns a singleton well formed node which is not a fragment.

On the other hand, an XPath such as `/PO/SHIPADDR/STATE/text()` would be considered a fragment, since it is not a well-formed XML document.

Querying XMLType Data using Text Operators

Oracle Text index works on CLOB and VARCHAR columns. It has been extended in Oracle to work on XMLType columns as well. The default behavior of Oracle Text index is to automatically create XML sections, when defined over XMLType columns. It also provides the CONTAINS operator which has been extended to support XPath.

Creating Text index over XMLType columns

Text index can be created by using the CREATE INDEX with the INDEXTYPE specification as with other CLOB or VARCHAR columns. However, since the XMLType is implemented as a virtual column, the text index is created using the functional index mechanism.

This requires that to create and use the text index in queries, in addition to having the privileges to create indexes and the privileges necessary to create text indexes, you would need to also need to have the privileges and settings necessary to create functional indexes. This includes

- QUERY_REWRITE privilege - You must have this privilege granted to create text indexes on XMLType columns in your own schema. If you need to create text indexes on XMLType columns in other schemas or on tables residing in other schemas, you must have the GLOBAL_QUERY_REWRITE privilege granted.
- QUERY_REWRITE_ENABLED parameter must be set to true.
- QUERY_REWRITE_INTEGRITY must be set to trusted for the queries to be rewritten to use the text index.

See Also:

- [Chapter 8, "Searching XML Data with Oracle Text"](#)
- *Oracle9i Text Reference*
- *Oracle9i Text Developer's Guide*

Differences between CONTAINS and ExistsNode/Extract

There are certain differences with regard to the XPath support inside CONTAINS and that supported through the ExistsNode and Extract functions.

- In this release, the XPath supported by the Oracle Text index is more powerful than the functional implementation, as it can satisfy certain equality predicates as well

- Since Oracle Text index ignores spaces, the XPath expression may not yield accurate results when spaces are significant.
- Oracle Text index also supports certain predicate expressions with string equality, but cannot support numerical and range comparisons.
- One other limitation is that the Oracle Text index may give wrong result if the XML document only has tag names and attribute names without any text. For example in the case of the following document,

```
<A>
  <B>
    <C>
  </C>
  </B>
  <D>
    <E>
  </E>
  </D>
</A>
```

the XPath expression - A/B/E will falsely match the above XML document.

- Both the functional and the Oracle Text index support navigation. Thus you can use the text index as a primary filter, to filter out all the documents that can potentially match the criterion in an efficient manner, and then apply secondary filters such as `existsNode()` or `extract()` operations on the remainder of the documents.

Indexing XMLType columns

We can create the following indexes using XMLType to speed up query evaluation.

- **Functional Indexes with XMLType.** Queries can be speeded up by building functional indexes on the `ExistsNode` or the `Extracted` portions of an XML document.

Example of Functional Indexes on Extract operation:

For instance to speed up the search on the query,

```
SELECT * FROM po_xml_tab e
WHERE e.poDoc.extract('//PONO/text()').getNumberVal()= 100;
```

we can create a functional index on the `Extract` function as:


```
CREATE INDEX city_index ON po_xml_tab
(poDoc.extract('//PONO/text()').getNumberVal());
```

With this index, the SQL query would use the functional index to evaluate the predicate instead of parsing the XML document per row and evaluating the XPath expression.

Example of Functional indexes on ExistsNode:

We can also create bitmapped functional indexes to speed up the evaluation of the operators. In particular, the ExistsNode is best suited, since it returns a value of 1 or 0 depending on whether the XPath is satisfied in the document or not.

For instance to speed up the query, that searches for whether the XML document contains an element called Shipping address at any level -

```
SELECT * FROM po_xml_tab e
WHERE e.poDoc.existsNode('//SHIPADDR') = 1;
```

we can create a bitmapped functional index on the ExistsNode function as:

```
CREATE INDEX po_index ON po_xml_tab
(poDoc.existsNode('//SHIPADDR'));
```

to speed up the query processing.

- Creating Text Indexes on XMLType Columns.** As explained earlier, you can create text indexes on the XMLType column. The index uses the PATH_SECTION_GROUP as the default section group when indexing XMLType columns. This is the default and can be overridden during index creation.

```
CREATE INDEX po_text_index ON
po_xml_tab(poDoc) indextype is ctxsys.context;
```

You can do text operations such as CONTAINS and SCORE.. on this XMLType column. In Oracle, the CONTAINS function has been enhanced to support XPath using two new operators, INPATH and HASPATH.

INPATH checks if the given word appears within the path specified and HASPATH checks if the given XPath is present in the document.

```
SELECT * FROM po_xml_doc w
WHERE CONTAINS(w.poDoc,
'haspath(/PO[./@CUSTNAME="John Nike"]') > 0;
```

Java Access to XMLType (oracle.xdb.XMLType)

XMLType can be accessed through the `oracle.xdb.XMLType` class in Java. The class provides functions similar to the XMLType in PL/SQL. The `oracle.xdb.XMLType` is a subclass of Oracle JDBC's `oracle.sql.OPAQUE` class.

Since the XMLType in the server is implemented as an opaque type, you would need to use the `getOPAQUE` call in JDBC to get the opaque type instance from the JDBC Resultset and then create an XMLType instance out of it. In future releases, JDBC would instantiate XMLTypes automatically.

You can bind XMLType to any XML data instance in JDBC using the `setObject` call in the `java.sql.PreparedStatement` interface.

The functions defined in the `oracle.xdb.XMLType` class help to retrieve the XML data in Java. Use the SQL functions to perform any queries etc.

XMLType Java Example 1: Selecting XMLType data in Java

You can select the XMLType in java in one of 2 ways,

- Use the `getClobVal()` or `getStringVal()` in SQL and get the result as a `oracle.sql.CLOB` or `java.lang.String` in Java. Here is a snippet of Java code that shows how to use this,

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci8:@", "scott", "tiger");

OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "select e.poDoc.getClobVal() poDoc, "+
           e.poDoc.getStringVal() poString "+
        " from po_xml_tab e");

ResultSet rset = stmt.executeQuery();
OracleResultSet orset = (OracleResultSet) rset;

// the first argument is a CLOB
oracle.sql.CLOB clb = orset.getCLOB(1);

// the second argument is a string..
String poString = orset.getString(2);
```

```
// now use the CLOB inside the program..
```

- Use the `getOPAQUE()` call in the `PreparedStatement` to get the whole `XMLType` instance and use the `XMLType` constructor to construct an `oracle.xdb.XMLType` class out of it. Then you can use the Java functions on the `XMLType` class to access the data.

```
import oracle.xdb.XMLType;
...

OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "select e.poDoc from po_xml_tab e");

ResultSet rset = stmt.executeQuery();
OracleResultSet orset = (OracleResultSet) rset;

// get the XMLType
XMLType poxml = XMLType(orset.getOPAQUE(1));

// get the XML as a string...
String poString = poxml.getStringVal();
```

XMLType Java Example 2: Updating XMLType data in Java

You can insert an `XMLType` in java in one of 2 ways,

- Bind a `CLOB` or a string to an insert/update/delete statement and use the `createXML()` constructor inside `SQL` to construct the `XML` instance,

```
OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "update po_xml_tab set poDoc = sys.XMLType.createXML(?) ");

// the second argument is a string..
String poString = "<PO><PONO>200</PONO><PNAME>PO_2</PNAME></PO>";

// now bind the string..
stmt.setString(1,poString);
stmt.execute();
```

- Use the `setObject()` (or `setOPAQUE()`) call in the `PreparedStatement` to set the whole `XMLType` instance.

```
import oracle.xdb.XMLType;
...
OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "update po_xml_tab set poDoc = ? ");

// the second argument is a string..
String poString = "<PO><PONO>200</PONO><PNAME>PO_2</PNAME></PO>";
XMLType poXML = XMLType.createXML(conn, poString);

// now bind the string..
stmt.setObject(1,poXML);
stmt.execute();
```

XMLType Java Example 3: Getting Metadata on XMLType

When selecting out XMLType values, JDBC describes the column as an OPAQUE type. You can select the column type name out and compare it with "XMLTYPE" to check if you are dealing with an XMLType,

```
import oracle.sql.*;
import oracle.jdbc.*;
...
OraclePreparedStatement stmt =
    (OraclePreparedStatement) conn.prepareStatement(
        "select poDoc from po_xml_tab");

OracleResultSet rset = (OracleResultSet)stmt.executeQuery();

// Now, we can get the resultset metadata
OracleResultSetMetaData mdata =
    (OracleResultSetMetaData)rset.getMetaData();

// Describe the column = the column type comes out as OPAQUE
// and column type name comes out as SYS.XMLTYPE
if (mdata.getColumnType(1) == OracleTypes.OPAQUE &&
    mdata.getColumnTypeName(1).compareTo("SYS.XMLTYPE") == 0)
{
    // we know it is an XMLtype..
}
```

Table 5–4 Summary of oracle.xdb.XMLType Member and Static Functions

Functions	Syntax Summary	Description
XMLType() (constructor)	PUBLIC oracle.xdb.XMLType(oracle.sql.OPAQUE opt)	Constructor to create the <code>oracle.xdb.XMLType</code> instance from an opaque instance. Currently, JDBC returns the XMLType data as an <code>oracle.sql.OPAQUE</code> instance. Use this constructor to construct an XMLType from the opaque instance. PARAMETERS: opq (IN) - A valid opaque instance.
createXML()	PUBLIC STATIC oracle.xdb.XMLType createXML(Connection conn, String xmlval)	Static function to create the <code>oracle.xdb.XMLType</code> instance from a string. Does not checks for well-formed XML value. Any database operation on the XML value would check for well-formedness. PARAMETERS: conn (IN) - A valid Oracle Connection xmlval (IN) - A Java string containing the XML value. RETURNS: An <code>oracle.xdb.XMLType</code> instance.
createXML()	PUBLIC STATIC oracle.xdb.XMLType createXML(Connection conn, oracle.sql.clob xmlVal)	Static function to create the <code>XMLType</code> instance from an <code>oracle.sql.CLOB</code> . Does not check for well-formedness. Any database operation would check for that. PARAMETERS: conn (IN) - A valid Oracle Connection. xmlval (IN) - CLOB containing the XML document RETURNS: An <code>oracle.xdb.XMLType</code> instance.
getClobVal()	PUBLIC oracle.sql.CLOB getClobVal()	Gets the document as a <code>oracle.sql.CLOB</code> . RETURNS: A CLOB containing the serialized XML representation. Free the temporary CLOB after use.
getStringVal()	PUBLIC java.lang.String getStringVal()	Gets the XML value as a string. RETURNS: A string containing the serialized XML representation, or in case of text nodes, the text itself.

XMLType Java Example 4: Updating an Element in XMLType Column

This example updates the “DISCOUNT” element inside PurchaseOrder stored in a XMLType column. It uses Java (JDBC) and the `oracle.xdb.XMLType` class. This

example also shows you how to insert/update/delete XMLTypes using Java (JDBC). It uses the parser to update an in-memory DOM tree and write the updated XML value to the column.

```
-- create po_xml_hist table to store old PurchaseOrders
create table po_xml_hist (
  xpo sys.xmltype
);

/*
  DESCRIPTION
    Example for oracle.xdb.XMLType

  NOTES
    Have classes12.zip, xmlparserv2.jar, and oraxdb.jar in CLASSPATH
*/

import java.sql.*;
import java.io.*;

import oracle.xml.parser.v2.*;
import org.xml.sax.*;
import org.w3c.dom.*;

import oracle.jdbc.driver.*;
import oracle.sql.*;

import oracle.xdb.XMLType;

public class tkxmtpje
{
  static String conStr = "jdbc:oracle:oci8:@";
  static String user = "scott";
  static String pass = "tiger";
  static String qryStr =
    "SELECT x.poDoc from po_xml_tab x "+
    "WHERE x.poDoc.extract('/PO/PONO/text()').getNumberVal()=200";

  static String updateXML(String xmlTypeStr)
  {
    System.out.println("\n=====");
    System.out.println("xmlType.getStringVal():");
  }
}
```

```

System.out.println(xmlTypeStr);
System.out.println("=====");
String outXML = null;
try{
    DOMParser parser = new DOMParser();
    parser.setValidationMode(false);
    parser.setPreserveWhitespace (true);

    parser.parse(new StringReader(xmlTypeStr));
    System.out.println("xmlType.getStringVal(): xml String is well-formed");

    XMLDocument doc = parser.getDocument();

    NodeList nl = doc.getElementsByTagName("DISCOUNT");

    for(int i=0;i<nl.getLength();i++){
        XMLElement discount = (XMLElement)nl.item(i);
        XMLNode textNode = (XMLNode)discount.getFirstChild();
        textNode.setNodeValue("10");
    }

    StringWriter sw = new StringWriter();
    doc.print(new PrintWriter(sw));

    outXML = sw.toString();

    //print modified xml
    System.out.println("\n=====");
    System.out.println("Updated PurchaseOrder:");
    System.out.println(outXML);
    System.out.println("=====");
}
catch ( Exception e )
{
    e.printStackTrace(System.out);
}
return outXML;
}

public static void main(String args[]) throws Exception
{
    try{

        System.out.println("qryStr="+ qryStr);

```

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci8:@", user, pass);

Statement s = conn.createStatement();
OraclePreparedStatement stmt;

ResultSet rset = s.executeQuery(qryStr);
OracleResultSet orset = (OracleResultSet) rset;

while(orset.next()){

//retrieve PurchaseOrder xml document from database
XMLType xt = XMLType.createXML(orset.getOPAQUE(1));

    //store this PurchaseOrder in po_xml_hist table
    stmt = (OraclePreparedStatement)conn.prepareStatement(
        "insert into po_xml_hist values(?)");

    stmt.setObject(1,xt); // bind the XMLType instance
    stmt.execute();

//update "DISCOUNT" element
    String newXML = updateXML(xt.getStringVal());

    // create a new instance of an XMLType from the updated value
    xt = XMLType.createXML(conn,newXML);

// update PurchaseOrder xml document in database
    stmt = (OraclePreparedStatement)conn.prepareStatement(
        "update po_xml_tab x set x.poDoc =? where "+
        "x.poDoc.extract('/PO/PONO/text()').getNumberVal(=)200");

    stmt.setObject(1,xt); // bind the XMLType instance
    stmt.execute();

    conn.commit();
    System.out.println("PurchaseOrder 200 Updated!");

}

//delete PurchaseOrder 1001
s.execute("delete from po_xml x "+
    "where x.xpo.extract"+
```



```

        ("/PurchaseOrder/PONO/text()").getNumberVal()=1001");
    System.out.println("PurchaseOrder 1001 deleted!");
}
catch( Exception e )
{
    e.printStackTrace(System.out);
}
}
}
}

```

```

-----
-- list PurchaseOrders
-----

```

```

set long 20000
set pages 100
select x.xpo.getClobVal()
from po_xml x;

```

Here is the resulting updated purchase order in XML:

```

<?xml version = '1.0'?>
<PurchaseOrder>
  <PONO>200</PONO>
  <CUSTOMER>
    <CUSTINO>2</CUSTINO>
    <CUSTNAME>John Nike</CUSTNAME>
    <ADDRESS>
      <STREET>323 College Drive</STREET>
      <CITY>Edison</CITY>
      <STATE>NJ</STATE>
      <ZIP>08820</ZIP>
    </ADDRESS>
    <PHONELIST>
      <VARCHAR2>609-555-1212</VARCHAR2>
      <VARCHAR2>201-555-1212</VARCHAR2>
    </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>20-APR-97</ORDERDATE>
  <SHIPDATE>20-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS>
    <LINEITEM_TYP LineItemNo="1">
      <ITEM StockNo="1004">
        <PRICE>6750</PRICE>
        <TAXRATE>2</TAXRATE>

```

```
</ITEM>
<QUANTITY>1</QUANTITY>
<DISCOUNT>10</DISCOUNT>
</LINEITEM_TYP>
<LINEITEM_TYP LineItemNo="2">
  <ITEM StockNo="1011">
    <PRICE>4500.23</PRICE>
    <TAXRATE>2</TAXRATE>
  </ITEM>
  <QUANTITY>2</QUANTITY>
  <DISCOUNT>10</DISCOUNT>
</LINEITEM_TYP>
</LINEITEMS>
<SHIPTOADDR>
  <STREET>55 Madison Ave</STREET>
  <CITY>Madison</CITY>
  <STATE>WI</STATE>
  <ZIP>53715</ZIP>
</SHIPTOADDR>
</PurchaseOrder>
```

Installing and using oracle.xdb.XMLType class

The oracle.xdb.XMLType is available in the **xdb_g.jar** file in the *ORACLE_HOME/rdbms/jlib* where *ORACLE_HOME* refers to the Oracle home directory.

Using oracle.xdb.XMLType inside JServer:

This class is pre-loaded in to the JServer and is available in the SYS schema.

It is not loaded however, if you have upgraded your database from an earlier version. If you need to upload the class into the JServer, you would need to run the *initxdbj.sql* file located in the *ORACLE_HOME/rdbms/admin* directory, while connected as SYS.

Using oracle.xdb.XMLType on the client:

If you need to use the oracle.xdb.XMLType class on the client side, then ensure that the xdb_g.jar file is listed in your CLASSPATH environment variable.

Native XML Generation

Oracle supports native XML generation with the following packages and functions:

- *DBMS_XMLGEN* PL/SQL supplied package. Gets XML from SQL queries. This is written in C and linked to the server for enhanced performance.
- SQL functions for getting XML from SQL queries are:
 - *SYS_XMLGEN* operates on rows, generating XML documents
 - *SYS_XMLAGG* operates on groups of rows, aggregating several XML documents into one

DBMS_XMLGEN

DBMS_XMLGEN creates XML documents from any SQL query by mapping the database query results into XML. It gets the XML document as a CLOB. It provides a “fetch” interface whereby you can specify the maximum rows and rows to skip. This is useful for pagination requirements in web applications. DBMS_XMLGEN also provides options for changing tag names for ROW, ROWSET, and so on.

The parameters of the package can restrict the number of rows retrieved, the enclosing tag names. To summarize, DBMS_XMLGEN PL/SQL package allows you:

- To create an XML document instance from any SQL query and get the document as a CLOB
- A “fetch” interface with maximum rows and rows to skip. For example, the first fetch could retrieve a maximum of 10 rows, skipping the first four. This is useful for pagination in web-based applications.
- Options for changing tag names for ROW, ROWSET, and so on.

See Also: ["Generating XML with XSU's OracleXMLQuery"](#) on page 7-2, in [Chapter 7, "XML SQL Utility \(XSU\)"](#), to compare the functionality OracleXMLQuery with DBMS_XMLGEN.

Sample Query Result

The following shows a sample result from executing the “select * from scott.emp” query on a database:

```
<?xml version="1.0"?>
<ROWSET>
```

```
<ROW>
  <EMPNO>30</EMPNO>
  <ENAME>Scott</ENAME>
  <SALARY>20000</SALARY>
</ROW>
<ROW>
  <EMPNO>30</EMPNO>
  <ENAME>Mary</ENAME>
  <AGE>40</AGE>
</ROW>
</ROWSET>
```

The result of the `getXML()` using `DBMS_XMLGen` package is a CLOB. The default mapping is as follows:

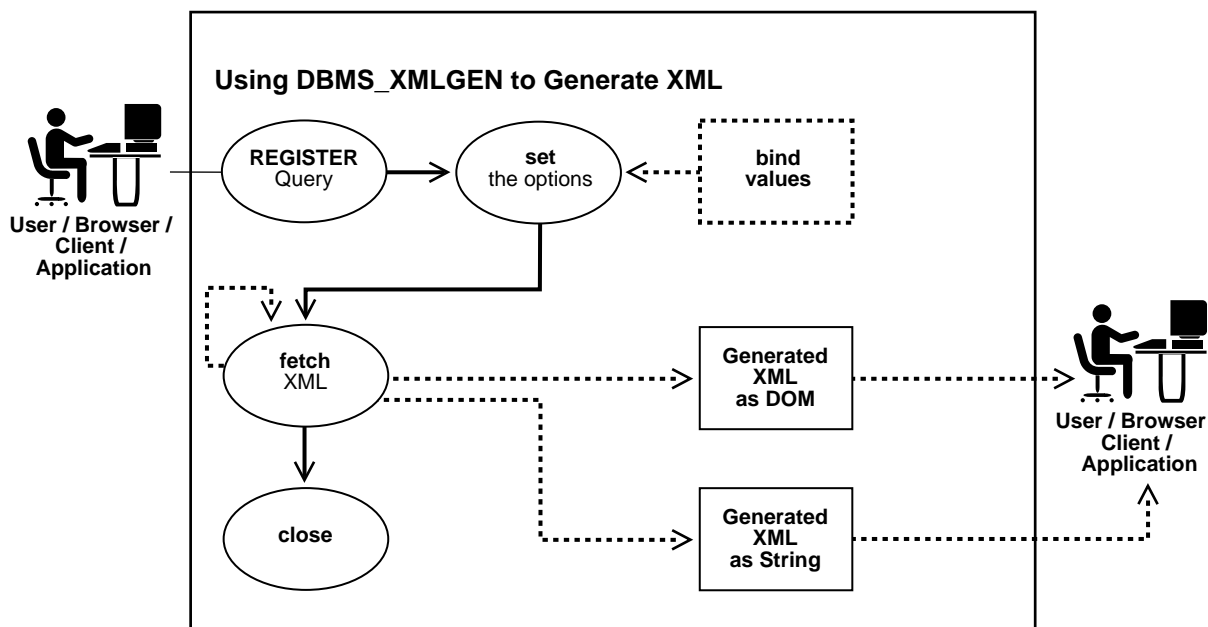
- Every row of the query result maps to an XML element with the default tag name “ROW”.
- The entire result is enclosed in a “ROWSET” element. These names are both configurable, using the `setRowTagName()` and `setRowSetTagName()` procedures in `DBMS_XMLGEN`.
- Each column in the SQL query result, maps as a subelement of the ROW element.
- All datatypes other than `CURSOR` expressions are supported by `DBMS_XMLGEN`. Binary data is transformed to its hexadecimal representation.

As the document is in a CLOB, it has the same encoding as the database character set. If the database character set is `SHIFTJIS`, then the XML document is `SHIFTJIS`.

DBMS_XMLGEN Calling Sequence

[Figure 5-2](#) summarizes the `DBMS_XMLGEN` calling sequence.

Figure 5-2 DBMS_XMLGEN Calling Sequence



Here is DBMS_XMLGEN's calling sequence:

1. Get the context from the package by supplying a SQL query and calling the `newContext()` call.
2. Pass the context to all the procedures/functions in the package to set the various options. For example to set the ROW element's name, use `setRowTag(ctx)`, where `ctx` is the context got from the previous `newContext()` call.
3. Get the XML result, using the `getXML()`. By setting the maximum rows to be retrieved per fetch using the `setMaxRows()` call, you can call this function repeatedly, getting the maximum number of row set per call. The function returns a null CLOB if there are no rows left in the query.

`getXML()` always returns an XML document, even if there were no rows to retrieve. If you want to know if there were any rows retrieved, use the function `getNumRowsProcessed()`.

4. You can reset the query to start again and repeat step 3.
5. Close the `closeContext()` to free up any resource allocated inside.

[Table 5–5](#) summarizes DBMS_XMLGEN functions and procedures.

Table 5–5 DBMS_XMLGEN Functions and Procedures

Function or Procedure	Description
DBMS_XMLGEN Type definitions SUBTYPE ctxHandle IS NUMBER	The context handle used by all functions. DTD or schema specifications: <ul style="list-style-type: none"> ■ NONE CONSTANT NUMBER:= 0; -- supported for this release. ■ DTD CONSTANT NUMBER:= 1; S ■ CHEMA CONSTANT NUMBER:= 2; Can be used in <code>getXML</code> function to specify whether to generate a DTD or XML Schema or none. Only the NONE specification is supported in the <code>getXML</code> functions for this release.
FUNCTION PROTOTYPES <code>newContext()</code>	Given a query string, generate a new context handle to be used in subsequent functions.
FUNCTION <code>newContext(queryString IN VARCHAR2)</code>	Returns a new context PARAMETERS: <code>queryString</code> (IN)- the query string, the result of which needs to be converted to XML RETURNS: Context handle. Call this function first to obtain a handle that you can use in the <code>getXML()</code> and other functions to get the XML back from the result.
<code>setRowTag()</code>	Sets the name of the element separating all the rows. The default name is ROW.
PROCEDURE <code>setRowTag(ctx IN ctxHandle, rowTag IN VARCHAR2);</code>	PARAMETERS: <code>ctx</code> (IN) - the context handle obtained from the <code>newContext</code> call, <code>rowTag</code> (IN) - the name of the ROW element. NULL indicates that you do not want the ROW element to be present. Call this function to set the name of the ROW element, if you do not want the default "ROW" name to show up. You can also set this to NULL to suppress the ROW element itself. Its an error if both the row and the rowset are null and there is more than one column or row in the output.

Table 5–5 DBMS_XMLGEN Functions and Procedures (Cont.)

Function or Procedure	Description
setRowSetTag()	Sets the name of the document's root element. The default name is "ROWSET"
PROCEDURE setRowSetTag(ctx IN ctxHandle, rowSetTag IN VARCHAR2);	PARAMETERS: ctx (IN) - the context handle obtained from the newContext call, rowsetTag (IN) - the name of the document element. NULL indicates that you do not want the ROW element to be present. Call this to set the name of the document root element, if you do not want the default "ROWSET" name in the output. You can also set this to NULL to suppress the printing of this element. However, this is an error if both the row and the rowset are null and there is more than one column or row in the output.
getXML()	Gets the XML document by fetching the maximum number of rows specified. It appends the XML document to the CLOB passed in.
PROCEDURE getXML(ctx IN ctxHandle, clobval IN OUT NCOPY clob, dtdOrSchema IN number:= NONE);	PARAMETERS: ctx (IN) - The context handle obtained from the newContext() call, clobval (IN/OUT) - the clob to which the XML document is to be appended, dtdOrSchema (IN) - whether we should generate the DTD or Schema. This parameter is NOT supported. Use this version of the getXML function, to avoid any extra CLOB copies and if you want to reuse the same CLOB for subsequent calls. This getXML call is more efficient than the next flavor, though this involves that you create the lob locator. When generating the XML, the number of rows indicated by the setSkipRows call are skipped, then 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 function to check if any rows were retrieved or not.
getXML()	Generates the XML document and return it as a CLOB.

Table 5-5 DBMS_XMLGEN Functions and Procedures (Cont.)

Function or Procedure	Description
FUNCTION getXML(ctx IN ctxHandle, dtdOrSchema IN number:= NONE) RETURN clob	PARAMETERS: ctx (IN) - The context handle obtained from the newContext() call, dtdOrSchema (IN) - whether we should generate the DTD or Schema. This parameter is NOT supported. RETURNS: A temporary CLOB containing the document. Free the temporary CLOB obtained from this function using the dbms_lob.freetemporary call.
FUNCTION getXMLType(ctx IN ctxHandle, dtdOrSchema IN number:= NONE) RETURN sys.XMLType	PARAMETERS: ctx (IN) - The context handle obtained from the newContext() call, dtdOrSchema (IN) - whether we should generate the DTD or Schema. This parameter is NOT supported. RETURNS: An XMLType instance containing the document.
getNumRowsProcessed()	Gets 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.
FUNCTION getNumRowsProcessed(ctx IN ctxHandle) RETURN number	PARAMETERS: queryString (IN)- the query string, the result of which needs to be converted to XML RETURNS: The number of rows processed in the last call to getXML. This does not include the number of rows skipped. Use this function to determine the terminating condition if you are calling getXML in a loop. Note that getXML would always generate a XML document even if there are no rows present.
setMaxRows()	Sets the maximum number of rows to fetch from the SQL query result for every invocation of the getXML call.
PROCEDURE setMaxRows(ctx IN ctxHandle, maxRows IN NUMBER);	PARAMETERS: ctx (IN) - the context handle corresponding to the query executed, maxRows (IN) - the maximum number of rows to get per call to getXML. The maxRows parameter can be used when generating paginated results using this utility. For instance when generating a page of XML or HTML data, you can restrict the number of rows converted to XML and then in subsequent calls, you can get the next set of rows and so on. This also can provide for faster response times.

Table 5-5 DBMS_XMLGEN Functions and Procedures (Cont.)

Function or Procedure	Description
setSkipRows()	Skips a given number of rows before generating the XML output for every call to the getXML routine.
PROCEDURE setSkipRows(ctx IN ctxHandle, skipRows IN NUMBER);	PARAMETERS: ctx (IN) - the context handle corresponding to the query executed, skipRows (IN) - the number of rows to skip per call to getXML. The skipRows parameter can be used when generating paginated results for stateless web pages using this utility. For instance when generating the first page of XML or HTML data, you can set skipRows to zero. For the next set, you can set the skipRows to the number of rows that you got in the first case.
setConvertSpecialChars()	Sets whether special characters in the XML data need to be converted into their escaped XML equivalent or not. For example, the "<" sign is converted to <. The default is to perform conversions.
PROCEDURE setConvertSpecialChars(ctx IN ctxHandle, conv IN boolean);	PARAMETERS: ctx (IN) - the context handle to use, conv (IN) - true indicates that conversion is needed. You can use this function to speed up the XML processing whenever you are sure that the input data cannot contain any special characters such as <, >, ", ' etc. which need to be escaped. Note that it is expensive to actually scan the character data to replace the special characters, particularly if it involves a lot of data. So in cases when the data is XML-safe, then this function can be called to improve performance.
useItemTagsForColl()	Sets the name of the collection elements. The default name for collection elements is the type name itself. You can override that to use the name of the column with the "_ITEM" tag appended to it using this function.
PROCEDURE useItemTagsForColl(ctx IN ctxHandle);	PARAMETERS: ctx (IN) - the context handle. If you have a collection of NUMBER, say, the default tag name for the collection elements is NUMBER. You can override this behavior and generate the collection column name with the _ITEM tag appended to it, by calling this procedure.
restartQuery()	Restarts the query and generate the XML from the first row again.

Table 5–5 DBMS_XMLGEN Functions and Procedures (Cont.)

Function or Procedure	Description
PROCEDURE restartQuery(ctx IN ctxHandle);	PARAMETERS: ctx (IN) - the context handle corresponding to the current query. You can call this to start executing the query again, without having to create a new context.
closeContext()	Closes a given context and releases all resources associated with that context, including the SQL cursor and bind and define buffers etc.
PROCEDURE closeContext(ctx IN ctxHandle);	PARAMETERS: ctx (IN) - the context handle to close. Closes all resources associated with this handle. After this you cannot use the handle for any other DBMS_XMLGEN function call.

DBMS_XMLGEN Example 1: Generating Simple XML

This example creates an XML document by selecting out the employee data from an object-relational table and puts the result CLOB into a table.

```
CREATE TABLE temp_clob_tab(result CLOB);

DECLARE
    qryCtx DBMS_XMLGEN.ctxHandle;
    result CLOB;
BEGIN
    qryCtx := dbms_xmlgen.newContext('SELECT * from scott.emp;');

    -- set the row header to be EMPLOYEE
    DBMS_XMLGEN.setRowTag(qryCtx, 'EMPLOYEE');

    -- now get the result
    result := DBMS_XMLGEN.getXML(qryCtx);

    INSERT INTO temp_clob_tab VALUES(result);
END;
/
```

Here is the XML generated from this example:

```
select * from temp_clob_tab;

RESULT
-----
<?xml version='1.0'?>
```

```

<ROWSET>
  <EMPLOYEE>
    <EMPNO>7369</EMPNO>
    <ENAME>SMITH</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>17-DEC-80</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </EMPLOYEE>
  <EMPLOYEE>
    <EMPNO>7499</EMPNO>
    <ENAME>ALLEN</ENAME>
    <JOB>SALESMAN</JOB>
    <MGR>7698</MGR>
    <HIREDATE>20-FEB-81</HIREDATE>
    <SAL>1600</SAL>
    <COMM>300</COMM>
    <DEPTNO>30</DEPTNO>
  </EMPLOYEE>
  . . .
</ROWSET>

```

DBMS_XMLGEN Example 2: Generating Simple XML with pagination

Instead of getting the whole XML for all the rows, we can use the “fetch” interface that the DBMS_XMLGEN provides to retrieve a fixed number of rows each time. This speeds up the response time and also can help in scaling applications which would need to use a DOM API over the result XML - particularly if the number of rows is large.

The following example illustrates how to use DBMS_XMLGEN to retrieve results from the scott.emp table:

```

-- create a table to hold the results..!
create table temp_clob_tab ( result clob);

declare
  qryCtx dbms_xmlgen.ctxHandle;
  result CLOB;
begin

  -- get the query context;
  qryCtx := dbms_xmlgen.newContext('select * from scott.emp');

```

```
-- set the maximum number of rows to be 5,
dbms_xmlgen.setMaxRows(qryCtx, 5);

loop
  -- now get the result
  result := dbms_xmlgen.getXML(qryCtx);

  -- if there were no rows processed, then quit..!
  exit when dbms_xmlgen.getNumRowsProcessed(qryCtx) = 0;

  -- do some processing with the lob data..!
  -- Here, we are inserting the results
  -- into a table. You can print the lob out, output it to a stream,
  -- put it in a queue
  -- or do any other processing.
  insert into temp_clob_tab values(result);

end loop;

end;
/
```

Here, for each set of 5 rows, we would get an XML document.

DBMS_XMLGEN Example 3: Generating Complex XML

Complex XML can be generated using Object types to represent nested structures

```
CREATE TABLE new_departments (
  department_id NUMBER PRIMARY KEY,
  department_name VARCHAR2(20)
);

CREATE TABLE new_employees (
  employee_id NUMBER PRIMARY KEY,
  last_name VARCHAR2(20),
  department_id NUMBER REFERENCES departments
);

CREATE TYPE emp_t AS OBJECT(
  "@employee_id" NUMBER,
  last_name VARCHAR2(20)
);

CREATE TYPE emplist_t AS TABLE OF emp_t;
```

```

CREATE TYPE dept_t AS OBJECT(
  "@department_id" NUMBER,
  department_name VARCHAR2(20),
  emplist emplist_t
);

qryCtx := dbms_xmlgen.newContext
('SELECT dept_t(department_id, department_name,
  CAST(MULTISET
    (SELECT e.employee_id, e.last_name
     FROM employees e
     WHERE e.department_id = d.department_id
     AS emplist_t)) AS deptxml
  FROM departments d');
DBMS_XMLGEN.setRowTag(qryCtx, NULL);

```

Here is the resulting XML:

```

<ROWSET>
  <DEPTXML DEPARTMENT_ID="10">
    <DEPARTMENT_NAME>SALES</DEPARTMENT_NAME>
    <EMPLIST>
      <EMP_T EMPLOYEE_ID="30">
        <LAST_NAME>Scott</LAST_NAME>
      </EMP_T>
      <EMP_T EMPLOYEE_ID="31">
        <LAST_NAME>Mary</LAST_NAME>
      </EMP_T>
    </EMPLIST>
  </DEPTXML>
  <DEPTXML DEPARTMENT_ID="20">
    ...
  </ROWSET>

```

Now, you can select the LOB data from the temp_clob_Tab table and verify the results. The result looks like the sample result shown in the previous section, ["Sample Query Result"](#) on page 5-41.

With relational data, the results are a flat non-nested XML document. To obtain *nested* XML structures, you can use object-relational data, where the mapping is as follows:

- **Object types** map as an XML element

- **Attributes of the type**, map to sub-elements of the parent element

Note: Complex structures can be obtained by using object types and creating object views or object tables. A canonical mapping is used to map object instances to XML.

The @ sign, when used in column or attribute names, is translated into an attribute of the enclosing XML element in the mapping.

DBMS_XMLGEN Example 4: Generating Complex XML #2 - Inputting User Defined Types To Get Nesting in XML Documents

When you input a user-defined type (UDT) value to DBMS_XMLGEN functions, the user-defined type gets mapped to an XML document using a canonical mapping. In the canonical mapping, user-defined type's *attributes* are mapped to XML *elements*.

Any attributes with names starting with "@" are mapped to an attribute of the preceding element.

User-defined types can be used to get nesting within the result XML document.

For example, consider the two tables, EMP and DEPT:

```
CREATE TABLE DEPT
(
  deptno number primary key,
  dname varchar2(20)
);

CREATE TABLE EMP
(
  empno number primary key,
  ename varchar2(20),
  deptno number references dept
);
```

Now, to generate a hierarchical view of the data, that is, departments with employees in them, you can define suitable object types to create the structure inside the database as follows:

```
CREATE TYPE EMP_T AS OBJECT
(
  "@empno" number, -- empno defined as an attribute!
  ename varchar2(20),
);
```

/
 You have defined the empno with an @ sign in front, to denote that it must be mapped as an attribute of the enclosing Employee element.

```
CREATE TYPE EMPLIST_T AS TABLE OF EMP_T;
/
CREATE TYPE DEPT_T AS OBJECT
(
  "@deptno" number,
  dname varchar2(20),
  emplist emplist_t
);
/
```

Department type, DEPT_T, represents the department as containing a list of employees. You can now query the employee and department tables and get the result as an XML document, as follows:

```
declare
  qryCtx dbms_xmlgen.ctxHandle;
  result CLOB;
begin
  -- get the query context;
  qryCtx := dbms_xmlgen.newContext(
    'SELECT
      dept_t(deptno,dname,
             CAST(MULTISET(select empno, ename
                           from emp e
                           where e.deptno = d.deptno) AS emplist_t)) AS deptxml
    FROM dept d');

  -- set the maximum number of rows to be 5,
  dbms_xmlgen.setMaxRows(qryCtx, 5);

  -- set no row tag for this result as we have a single ADT column
  dbms_xmlgen.setRowTag(qryCtx,null);

  loop
    -- now get the result
    result := dbms_xmlgen.getXML(qryCtx);

    -- if there were no rows processed, then quit..!
    exit when dbms_xmlgen.getNumRowsProcessed(qryCtx) = 0;
```

```
        -- do whatever with the result..!  
    end loop;  
end;  
/
```

The **MULTISET** operator treats the result of the subset of employees working in the department as a list and the **CAST** around it, cast's it to the appropriate collection type. You then create a department instance around it and call the **DBMS_XMLGEN** routines to create the XML for the object instance. The result is:

```
<?xml version="1.0"?>  
<ROWSET>  
  <DEPTXML deptno="10">  
    <DNAME>Sports</DNAME>  
    <EMPLIST>  
      <EMP_T empno="200">  
        <ENAME>John</ENAME>  
      </EMP_T>  
      <EMP_T empno="300">  
        <ENAME>Jack</ENAME>  
      </EMP_T>  
    </EMPLIST>  
  </DEPTXML>  
  <DEPTXML deptno="20">  
    <!-- .. other columns -->  
  </DEPTXML>  
</ROWSET>
```

The default name "ROW" is not present because you set that to NULL. The deptno and empno have become attributes of the enclosing element.

DBMS_XMLGEN Example 5: Generating a Purchase Order From the Database in XML Format

This example uses `DBMS_XMLGEN.getXMLType()` to generate PurchaseOrder in XML format from a relational database using object views.

```
-----  
-- Create relational schema and define Object Views  
-- Note: DBMS_XMLGEN Package maps UDT attributes names  
--       starting with '@' to xml attributes  
-----  
-- Purchase Order Object View Model
```



```
-- PhoneList Varray object type
CREATE TYPE PhoneList_vartyp AS VARRAY(10) OF VARCHAR2(20)
/

-- Address object type
CREATE TYPE Address_typ AS OBJECT (
  Street      VARCHAR2(200),
  City        VARCHAR2(200),
  State       CHAR(2),
  Zip         VARCHAR2(20)
)
/

-- Customer object type
CREATE TYPE Customer_typ AS OBJECT (
  CustNo      NUMBER,
  CustName    VARCHAR2(200),
  Address     Address_typ,
  PhoneList   PhoneList_vartyp
)
/

-- StockItem object type
CREATE TYPE StockItem_typ AS OBJECT (
  "@StockNo"  NUMBER,
  Price       NUMBER,
  TaxRate     NUMBER
)
/

-- LineItems object type
CREATE TYPE LineItem_typ AS OBJECT (
  "@LineItemNo" NUMBER,
  Item        StockItem_typ,
  Quantity    NUMBER,
  Discount    NUMBER
)
/

-- LineItems Nested table
CREATE TYPE LineItems_ntabtyp AS TABLE OF LineItem_typ
/

-- Purchase Order object type
CREATE TYPE PO_typ AUTHID CURRENT_USER AS OBJECT (
  PONO        NUMBER,
```

```
    Cust_ref          REF Customer_typ,
    OrderDate         DATE,
    ShipDate          TIMESTAMP,
    LineItems_ntab    LineItems_ntabtyp,
    ShipToAddr        Address_typ
  )
/

-- Create Purchase Order Relational Model tables

--Customer table
CREATE TABLE Customer_tab(
  CustNo             NUMBER NOT NULL,
  CustName           VARCHAR2(200) ,
  Street             VARCHAR2(200) ,
  City               VARCHAR2(200) ,
  State              CHAR(2) ,
  Zip                VARCHAR2(20) ,
  Phone1             VARCHAR2(20),
  Phone2             VARCHAR2(20),
  Phone3             VARCHAR2(20),
  constraint cust_pk PRIMARY KEY (CustNo)
)
ORGANIZATION INDEX OVERFLOW;

-- Purchase Order table
CREATE TABLE po_tab (
  PONo              NUMBER, /* purchase order no */
  Custno            NUMBER constraint po_cust_fk references Customer_tab,
                    /* Foreign KEY referencing customer */
  OrderDate         DATE, /* date of order */
  ShipDate          TIMESTAMP, /* date to be shipped */
  ToStreet          VARCHAR2(200), /* shipto address */
  ToCity            VARCHAR2(200),
  ToState           CHAR(2),
  ToZip             VARCHAR2(20),
  constraint po_pk PRIMARY KEY(PONo)
);

--Stock Table
CREATE TABLE Stock_tab (
  StockNo           NUMBER constraint stock_uk UNIQUE,
  Price             NUMBER,
  TaxRate           NUMBER
);
```

```

--Line Items Table
CREATE TABLE LineItems_tab(
  LineItemNo      NUMBER,
  PONo            NUMBER constraint LI_PO_FK REFERENCES po_tab,
  StockNo        NUMBER ,
  Quantity       NUMBER,
  Discount       NUMBER,
  constraint LI_PK PRIMARY KEY (PONo, LineItemNo)
);

-- create Object Views

--Customer Object View
CREATE OR REPLACE VIEW Customer OF Customer_typ
  WITH OBJECT IDENTIFIER(CustNo)
  AS SELECT c.Custno, C.custname,
           Address_typ(C.Street, C.City, C.State, C.Zip),
           PhoneList_vartyp(Phone1, Phone2, Phone3)
  FROM Customer_tab c;

--Purchase order view
CREATE OR REPLACE VIEW PO OF PO_typ
  WITH OBJECT IDENTIFIER (PONo)
  AS SELECT P.PONo,
           MAKE_REF(Customer, P.Custno),
           P.OrderDate,
           P.ShipDate,
           CAST( MULTISSET(
             SELECT LineItem_typ( L.LineItemNo,
                                StockItem_typ(L.StockNo,S.Price,S.TaxRate),
                                L.Quantity, L.Discount)
             FROM LineItems_tab L, Stock_tab S
             WHERE L.PONo = P.PONo and S.StockNo=L.StockNo )
             AS LineItems_ntabtyp),
           Address_typ(P.ToStreet,P.ToCity, P.ToState, P.ToZip)
  FROM PO_tab P;

-- create table with XMLType column to store po in XML format
create table po_xml_tab(
  poid number,
  poDoc SYS.XMLType /* purchase order in XML format */
)
/

```

```
-----  
-- Populate data  
-----  
-- Establish Inventory  
  
INSERT INTO Stock_tab VALUES(1004, 6750.00, 2) ;  
INSERT INTO Stock_tab VALUES(1011, 4500.23, 2) ;  
INSERT INTO Stock_tab VALUES(1534, 2234.00, 2) ;  
INSERT INTO Stock_tab VALUES(1535, 3456.23, 2) ;  
  
-- Register Customers  
  
INSERT INTO Customer_tab  
VALUES (1, 'Jean Nance', '2 Avocet Drive',  
        'Redwood Shores', 'CA', '95054',  
        '415-555-1212', NULL, NULL) ;  
  
INSERT INTO Customer_tab  
VALUES (2, 'John Nike', '323 College Drive',  
        'Edison', 'NJ', '08820',  
        '609-555-1212', '201-555-1212', NULL) ;  
  
-- Place Orders  
  
INSERT INTO PO_tab  
VALUES (1001, 1, '10-APR-1997', '10-MAY-1997',  
        NULL, NULL, NULL, NULL) ;  
  
INSERT INTO PO_tab  
VALUES (2001, 2, '20-APR-1997', '20-MAY-1997',  
        '55 Madison Ave', 'Madison', 'WI', '53715') ;  
  
-- Detail Line Items  
  
INSERT INTO LineItems_tab VALUES(01, 1001, 1534, 12, 0) ;  
INSERT INTO LineItems_tab VALUES(02, 1001, 1535, 10, 10) ;  
INSERT INTO LineItems_tab VALUES(01, 2001, 1004, 1, 0) ;  
INSERT INTO LineItems_tab VALUES(02, 2001, 1011, 2, 1) ;  
  
-----  
-- Use DBMS_XMLGEN Package to generate PO in XML format  
-- and store SYS.XMLType in po_xml table  
-----
```

```

declare
  qryCtx dbms_xmlgen.ctxHandle;
  pxml SYS.XMLType;
  cxml clob;
begin

  -- get the query context;
  qryCtx := dbms_xmlgen.newContext('
        select pono,deref(cust_ref) customer,p.OrderDate,p.shipdate,
               lineitems_ntab lineitems,shiptoaddr
        from po p'
        );

  -- set the maximum number of rows to be 1,
  dbms_xmlgen.setMaxRows(qryCtx, 1);
  -- set rowset tag to null and row tag to PurchaseOrder
  dbms_xmlgen.setRowSetTag(qryCtx,null);
  dbms_xmlgen.setRowTag(qryCtx, 'PurchaseOrder');

  loop
    -- now get the po in xml format
    pxml := dbms_xmlgen.getXMLType(qryCtx);

    -- if there were no rows processed, then quit..!
    exit when dbms_xmlgen.getNumRowsProcessed(qryCtx) = 0;

    -- Store SYS.XMLType po in po_xml table (get the pono out)
    insert into po_xml_tab (poid, poDoc)
      values(
        pxml.extract('//PONO/text()').getNumberVal(),
        pxml);
  end loop;
end;
/

-----
-- list xml PurchaseOrders
-----

set long 100000
set pages 100
select x.xpo.getClobVal() xpo
from   po_xml x;

PurchaseOrder 1001:

```

This produces the following purchase order XML document:

```
<?xml version="1.0"?>
<PurchaseOrder>
  <PONO>1001</PONO>
  <CUSTOMER>
    <CUSTNO>1</CUSTNO>
    <CUSTNAME>Jean Nance</CUSTNAME>
    <ADDRESS>
      <STREET>2 Avocet Drive</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
      <ZIP>95054</ZIP>
    </ADDRESS>
    <PHONELIST>
      <VARCHAR2>415-555-1212</VARCHAR2>
    </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>10-APR-97</ORDERDATE>
  <SHIPDATE>10-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS>
    <LINEITEM_TYP LineItemNo="1">
      <ITEM StockNo="1534">
        <PRICE>2234</PRICE>
        <TAXRATE>2</TAXRATE>
      </ITEM>
      <QUANTITY>12</QUANTITY>
      <DISCOUNT>0</DISCOUNT>
    </LINEITEM_TYP>
    <LINEITEM_TYP LineItemNo="2">
      <ITEM StockNo="1535">
        <PRICE>3456.23</PRICE>
        <TAXRATE>2</TAXRATE>
      </ITEM>
      <QUANTITY>10</QUANTITY>
      <DISCOUNT>10</DISCOUNT>
    </LINEITEM_TYP>
  </LINEITEMS>
  <SHIPTOADDR/>
</PurchaseOrder>
```

PurchaseOrder 2001:

```
<?xml version="1.0"?>
```

```
<PurchaseOrder>
  <PONO>2001</PONO>
  <CUSTOMER>
    <CUSTNO>2</CUSTNO>
    <CUSTNAME>John Nike</CUSTNAME>
    <ADDRESS>
      <STREET>323 College Drive</STREET>
      <CITY>Edison</CITY>
      <STATE>NJ</STATE>
      <ZIP>08820</ZIP>
    </ADDRESS>
    <PHONELIST>
      <VARCHAR2>609-555-1212</VARCHAR2>
      <VARCHAR2>201-555-1212</VARCHAR2>
    </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>20-APR-97</ORDERDATE>
  <SHIPDATE>20-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS>
    <LINEITEM_TYP LineItemNo="1">
      <ITEM StockNo="1004">
        <PRICE>6750</PRICE>
        <TAXRATE>2</TAXRATE>
      </ITEM>
      <QUANTITY>1</QUANTITY>
      <DISCOUNT>0</DISCOUNT>
    </LINEITEM_TYP>
    <LINEITEM_TYP LineItemNo="2">
      <ITEM StockNo="1011">
        <PRICE>4500.23</PRICE>
        <TAXRATE>2</TAXRATE>
      </ITEM>
      <QUANTITY>2</QUANTITY>
      <DISCOUNT>1</DISCOUNT>
    </LINEITEM_TYP>
  </LINEITEMS>
  <SHIPTOADDR>
    <STREET>55 Madison Ave</STREET>
    <CITY>Madison</CITY>
    <STATE>WI</STATE>
    <ZIP>53715</ZIP>
  </SHIPTOADDR>
</PurchaseOrder>
```

SYS_XMLGEN

Oracle introduces a new SQL function, `SYS_XMLGEN()`, to generate XML in SQL queries. `DBMS_XMLGEN` and other packages operate at a query level, giving aggregated results for the *entire* query. `SYS_XMLGEN` takes in a *single* argument in an SQL query and converts it (the result) to XML.

`SYS_XMLGEN` takes a scalar value, object type, or `XMLType` instance to be converted to an XML document. It also takes an optional `XMLGenFormatType` object that you can use to specify formatting options for the resulting XML document.

`SYS_XMLGEN` returns a `XMLType`.

It is used to create and query XML instances within SQL queries, as follows:

```
SQL> SELECT SYS_XMLGEN(employee_id)
         2 FROM employees WHERE last_name LIKE
         'Scott%';
```

The resulting XML document is:

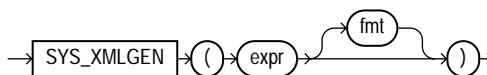
```
<?xml version='1.0'?>
<employee_id>60</employee_id>
```

SYS_XMLGEN Syntax

The `SYS_XMLGEN` function takes an expression that evaluates to a particular row and column of the database, and returns an instance of type `XMLType` containing an XML document. See [Figure 5-3](#). The `expr` can be a scalar value, a user-defined type, or a `XMLType` instance.

- If `expr` is a scalar value, the function returns an XML element containing the scalar value.
- If `expr` is a type, the function maps the user-defined type attributes to XML elements.
- If `expr` is a `XMLType` instance, then the function encloses the document in an XML element whose default tag name is `ROW`.

By default the elements of the XML document match the elements of `expr`. For example, if `expr` resolves to a column name, the enclosing XML element will be the same column name. If you want to format the XML document differently, specify `fmt`, which is an instance of the `XMLGenFormatType` object.

Figure 5–3 SYS_XMLGEN Syntax

The following example retrieves the employee email ID from the sample table `oe.employees` where the `employee_id` value is 205, and generates an instance of a `XMLType` containing an XML document with an `EMAIL` element.

```

SELECT SYS_XMLGEN(email).getStringVal()
       FROM employees
       WHERE employee_id = 205;

```

```

SYS_XMLGEN(EMAIL).GETSTRINGVAL()
-----

```

```
<EMAIL>SHIGGENS</EMAIL>
```

Why is SYS_XMLGEN so Powerful?

`SYS_XMLGEN()` is powerful for the following reasons:

- You can create and query XML instances *within* SQL queries.
- Using the object-relational infrastructure, you can create complex and nested XML instances from simple relational tables.

`SYS_XMLGEN()` creates an XML document from either of the following:

- A user-defined type (UDT) instance
- A scalar value passed

and returns an `XMLType` instance contained in the document.

`SYS_XMLGEN()` also optionally inputs a `XMLGenFormatType` object type through which you can customize the SQL results. A `NULL` format object implies that the default mapping behavior is to be used.

XMLGenFormatType Object

`XMLGenFormatType` can be used to specify formatting arguments to `SYS_XMLGEN` and `SYS_XMLAGG` functions. [Table 5–6](#) lists the `XMLGenFormatType` attributes.

Table 5–6 XMLGenFormatType Attributes

XMLGenFormatType Attribute	Description
enclTag	The name of the enclosing tag to use
schemaType	Not currently supported, but in future will be used to specify how the XMLSchema should be generated.
processingIns	Can specify processing instructions such as stylesheet instructions that need to be appended to the beginning of the document.
dburl	Not used currently.
targetNameSpace	Not used currently, but will be used for XMLSchema generation.

Creating a Formatting Object with createFormat

You can use the static member function *createFormat* to create a formatting object. This function has most of the values defaulted. For example:

```
-- function to create the formatting object..
STATIC MEMBER FUNCTION createFormat(
    enclTag IN varchar2 := null,
    schemaType IN varchar2 := 'NO_SCHEMA'
    schemaName IN varchar2 := null,
    targetNameSpace IN varchar2 := null,
    dburl IN varchar2 := null,
    processingIns IN varchar2 := null)
RETURN XMLGenFormatType;
```

SYS_XMLGEN Example 1: Converting a Scalar Value to an XML Document Element's Contents

When you input a scalar value to *SYS_XMLGEN()*, *SYS_XMLGEN()* converts the scalar value to an *element* containing the scalar value. For example:

```
select sys_xmlgen(empno) from scott.emp where rownum < 1;
```

returns an XML document that contains the empno value as an element, as follows:

```
<?xml version="1.0"?>
<EMPNO>30</EMPNO>
```

The enclosing element name, in this case EMPNO, is derived from the column name passed to the operator. Also, note that the result of the SELECT statement is a row containing a XMLType.

Note: Currently, SQL*Plus cannot display XMLType properly, so you need to extract the LOB value out and display that. Use the *getClobval()* function on the XMLType to retrieve the CLOB value. For example,

```
SELECT sys_xmlgen(empno).getclobval() FROM scott.emp
WHERE rownum < 1;
```

In the last example, you used the column name “EMPNO” for the document. If the column name cannot be derived directly, then the default name “ROW” is used. For example, in the following case:

```
select sys_xmlgen(empno).getClobVal()  
from scott.emp  
where rownum < 1;
```

you get the following XML output:

```
<?xml version="1.0"?>  
<ROW>60</ROW>
```

since the function cannot infer the name of the expression. You can override the default ROW tag by supplying an `XMLGenFormatType` object to the first argument of the operator.

For example, in the last case, if you wanted the result to have EMPNO as the tag name, you can supply a formatting argument to the function, as follows:

```
select sys_xmlgen(empno *2,  
                sys.xmlgenformattype.createformat('EMPNO')).getClobVal()  
from dual;
```

This results in the following XML:

```
<?xml version="1.0"?>  
<EMPNO>60</EMPNO>
```

Note: Currently, CURSOR expressions are not supported as input values.

SYS_XMLGEN Example 2: Converting a User-Defined Type (UDT) to XML

When you input a user-defined type (UDT) value to `SYS_XMLGEN()`, the user-defined type gets mapped to an XML document using a canonical mapping. In the canonical mapping the user-defined type's attributes are mapped to XML elements.

Any type attributes with names starting with "@" are mapped to an attribute of the preceding element. User-defined types can be used to get nesting within the result XML document.

Using the same example as given in the `DBMS_XMLGEN` section ("[DBMS_XMLGEN Example 4: Generating Complex XML #2 - Inputting User Defined Types To Get Nesting in XML Documents](#)" on page 5-52), you can generate a hierarchical XML for the employee, department example as follows:

```

SELECT SYS_XMLGEN(
  dept_t(deptno,dname,
    CAST(MULTISET(
      select empno, ename
      from emp e
      where e.deptno = d.deptno) AS emplist_t))) .getClobVal()
  AS deptxml
FROM dept d;

```

The **MULTISET** operator treats the result of the subset of employees working in the department as a list and the **CAST** around it, cast's it to the appropriate collection type. You then create a department instance around it and call **SYS_XMLGEN()** to create the XML for the object instance.

The result is:

```

<?xml version="1.0"?>
<ROW DEPTNO="100">
  <DNAME>Sports</DNAME>
  <EMPLIST>
    <EMP_T EMPNO="200">
      <ENAME>John</ENAME>
    <EMP_T>
    <EMP_T>
      <ENAME>Jack</ENAME>
    </EMP_T>
  </EMPLIST>
</ROW>

```

per row of the department. The default name “ROW” is present because the function cannot deduce the name of the input operand directly.

The difference between the **SYS_XMLGEN** and the **DBMS_XMLGEN** is apparent from this example:

- **SYS_XMLGEN** works inside SQL queries and operates on the expressions and columns within the row
- **DBMS_XMLGEN** works on the entire result set

SYS_XMLGEN Example 3: Converting XMLType Instance

If you pass an XML document into **SYS_XMLGEN()**, **SYS_XMLGEN** encloses the document (or fragment) with an element, whose tag name is the default “ROW”, or

the name passed in through the formatting object. This functionality can be used to turn document fragments into well formed documents.

For example, the extract operation on the following document, can return a fragment. If you extract out the EMPNO elements from the following document:

```
<DOCUMENT>
  <EMPLOYEE>
    <ENAME>John</ENAME>
    <EMPNO>200</EMPNO>
  </EMPLOYEE>
  <EMPLOYEE>
    <ENAME>Jack</ENAME>
    <EMPNO>400</EMPNO>
  </EMPLOYEE>
  <EMPLOYEE>
    <ENAME>Joseph</ENAME>
    <EMPNO>300</EMPNO>
  </EMPLOYEE>
</DOCUMENT>
```

Using the following statement:

```
select e.xmldoc.extract('/DOCUMENT/EMPLOYEE/ENAME')
       from po_xml_tab e;
```

you get a document fragment such as the following:

```
<ENAME>John</ENAME>
<ENAME>Jack</ENAME>
<ENAME>Joseph</ENAME>
```

You can make this fragment a valid XML document, by calling `SYS_XMLGEN()` to put an enclosing element around the document, as follows:

```
select SYS_XMLGEN(e.xmldoc.extract('/DOCUMENT/EMPLOYEE/ENAME')).getclobval()
       from po_xml_tab e;
```

This places an element “ROW” around the result, as follows:

```
<?xml version="1.0"?>
<ROW>
  <ENAME>John</ENAME>
  <ENAME>Jack</ENAME>
  <ENAME>Joseph</ENAME>
</ROW>
```

Note: If the input was a column, then the column name would have been used as default. You can override the enclosing element name using the formatting object that can be passed in as an additional argument to the function.

SYS_XMLGEN() Example 4: Using SYS_XMLGEN() with Object Views

```
-- create Purchase Order object type
CREATE OR REPLACE TYPE PO_typ AUTHID CURRENT_USER AS OBJECT (
  PONO                NUMBER,
  Customer            Customer_typ,
  OrderDate          DATE,
  ShipDate           TIMESTAMP,
  LineItems_ntab     LineItems_ntabtyp,
  ShipToAddr         Address_typ
)
/

--Purchase order view
CREATE OR REPLACE VIEW PO OF PO_typ
WITH OBJECT IDENTIFIER (PONO)
AS SELECT P.PONo,
        Customer_typ(P.Custno,C.CustName,C.Address,C.PhoneList),
        P.OrderDate,
        P.ShipDate,
        CAST( MULTISSET(
            SELECT LineItem_typ( L.LineItemNo,
                                StockItem_typ(L.StockNo,S.Price,S.TaxRate),
                                L.Quantity, L.Discount)
            FROM LineItems_tab L, Stock_tab S
            WHERE L.PONo = P.PONo and S.StockNo=L.StockNo )
        AS LineItems_ntabtyp),
        Address_typ(P.ToStreet,P.ToCity, P.ToState, P.ToZip)
FROM PO_tab P, Customer C
WHERE P.CustNo=C.custNo;

-----
-- Use SYS_XMLGEN() to generate PO in XML format
-----

set long 20000
set pages 100
SELECT SYS_XMLGEN(value(p),
                  sys.xmlgenformatType.createFormat('PurchaseOrder')).getClobVal()
```

```
PO
FROM po p
WHERE p.pono=1001;
```

This returns the Purchase Order in XML format:

```
<?xml version="1.0"?>
<PurchaseOrder>
  <PONO>1001</PONO>
  <CUSTOMER>
    <CUSTNO>1</CUSTNO>
    <CUSTNAME>Jean Nance</CUSTNAME>
    <ADDRESS>
      <STREET>2 Avocet Drive</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
      <ZIP>95054</ZIP>
    </ADDRESS>
    <PHONELIST>
      <VARCHAR2>415-555-1212</VARCHAR2>
    </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>10-APR-97</ORDERDATE>
  <SHIPDATE>10-MAY-97 12.00.00.000000 AM</SHIPDATE>
  <LINEITEMS_NTAB>
    <LINEITEM_TYP LineItemNo="1">
      <ITEM StockNo="1534">
        <PRICE>2234</PRICE>
        <TAXRATE>2</TAXRATE>
      </ITEM>
      <QUANTITY>12</QUANTITY>
      <DISCOUNT>0</DISCOUNT>
    </LINEITEM_TYP>
    <LINEITEM_TYP LineItemNo="2">
      <ITEM StockNo="1535">
        <PRICE>3456.23</PRICE>
        <TAXRATE>2</TAXRATE>
      </ITEM>
      <QUANTITY>10</QUANTITY>
      <DISCOUNT>10</DISCOUNT>
    </LINEITEM_TYP>
  </LINEITEMS_NTAB>
  <SHIPTOADDR/>
</PurchaseOrder>
```

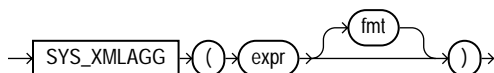

SYS_XMLAGG

`SYS_XMLAGG` aggregates all input documents and produces a single XML document. It aggregates (concatenates) fragments. `SYS_XMLAGG` takes in an `XMLType` as argument, and aggregates (or concatenates) all XML documents in rows into a single document per group. Use `SYS_XMLAGG()` for either of the following tasks:

- Aggregate fragments together
- Aggregate related XML data

In [Figure 5-4](#) shows how `SYS_XMLAGG` function aggregates all XML documents or fragments, represented by `expr`, and produces a single XML document. It adds a new enclosing element with a default name `ROWSET`. To format the XML document differently, specify `fmt`, which is an instance of the `SYS.XMLGenFormatType` object.

Figure 5-4 `SYS_XMLAGG` Syntax



For example:

```

SQL> SELECT SYS_XMLAGG(SYS_XMLGEN(last_name)
2      ,      SYS.XMLGENFORMATTYPE.createFormat
3      ('EmployeeGroup')).getClobVal()
4 FROM employees
5 GROUP BY department_id;
  
```

This generates the following XML document:

```

<EmployeeGroup>
  <last_name>Scott</last_name>
  <last_name>Mary</last_name>
</EmployeeGroup >
<EmployeeGroup >
  <last_name>Jack</last_name>
  <last_name>John</last_name>
</EmployeeGroup >
  
```

SYS_XMLAGG Example 1: Aggregating XML from Relational Data

Consider the SELECT statement:

```
SELECT SYS_XMLAGG(SYS_XMLGEN(ename)).getClobVal() xml_val
FROM scott.emp
GROUP BY deptno;
```

This returns the following XML document:

```
xml_val
-----
<ROWSET>
  <ENAME>John</ENAME>
  <ENAME>Jack</ENAME>
  <ENAME>Joseph</ENAME>
</ROWSET>

<ROWSET>
  <ENAME>Scott</ENAME>
  <ENAME>Adams</ENAME>
</ROWSET>
```

2 rows selected

Here, you grouped all the employees belonging to a particular department and aggregated all the XML documents produced by the SYS_XMLGEN function. In the previous SQL statement, if you wanted to enclose the result of each group in an EMPLOYEE tag, use the following statement:

```
select sys_xmlagg(sys_xmlgen(ename),
  sys.xmlgenformattype.createformat('EMPLOYEE')).getClobVal() xmlval
  from scott.emp
  group by deptno);
```

This returns the following:

```
xml_val
-----
<EMPLOYEE>
  <ENAME>John</ENAME>
  <ENAME>Jack</ENAME>
  <ENAME>Joseph</ENAME>
</EMPLOYEE>

<EMPLOYEE>
  <ENAME>Scott</ENAME>
```

```

    <ENAME>Adams</ENAME>
  </EMPLOYEE>

```

2 rows selected

SYS_XMLAGG Example 2: Aggregating XMLType Instances Stored in Tables

You can also aggregate XMLType instances that are stored in tables or selected out from functions. Assuming that you have a table defined as follows:

```

CREATE TABLE po_tab
(
  pono number primary key,
  orderdate date,
  poxml sys.XMLType;
);

insert into po_Tab values (100,'10-11-2000',
  '<?xml version="1.0"?><PO pono="100"><PONAME>Po_1</PONAME></PO>');
insert into po_Tab values (200,'10-23-1999',
  '<?xml version="1.0"?><PO pono="200"><PONAME>Po_2</PONAME></PO>');

```

You can now aggregate the purchase orders into a single purchase order using the SYS_XMLAGG function as follows:

```

select SYS_XMLAGG(poxml,sys.xmlgenformattype.createformat('POSET'))
from po_Tab;

```

This produces a single XML document of the form:

```

<?xml version="1.0"?>
<POSET>
  <PO pono="100">
    <PONAME>Po_1</PONAME>
  </PO>
  <PO pono="200">
    <PONAME>Po_2</PONAME>
  </PO>
</POSET>

```

SYS_XMLAGG Example 3: Aggregating XMLType Fragments

The Extract() function in XMLType, allows you to extract fragments of XML documents. You can also aggregate these fragments together using the SYS_XMLAGG function. For example, from the previous example, if you extract out the

PONAME elements alone, you can aggregate those together using the `SYS_XMLAGG` function as follows:

```
select SYS_XMLAGG(p.po_xml.extract('//PONAME')).getclobval()
from po_tab p;
```

This produces the following XML document:

```
<?xml version="1.0"?>
<ROWSET>
  <PONAME>Po_1</PONAME>
  <PONAME>Po_2</PONAME>
</ROWSET>
```

SYS_XMLAGG Example 4: Aggregating all Purchase Orders into One XML Document

```
set long 20000
set pages 200
SELECT SYS_XMLAGG(SYS_XMLGEN(value(p),
                             sys.xmlgenformatType.createFormat('PurchaseOrder'))).getClobVal()
PO
FROM po p;
```

This returns all Purchase Orders in one XML Document, namely that enclosed in the ROWSET element:

```
<?xml version="1.0"?>
<ROWSET>
<PurchaseOrder>
  <PONO>1001</PONO>
  <CUSTOMER>
    <CUSTNO>1</CUSTNO>
    <CUSTNAME>Jean Nance</CUSTNAME>
    <ADDRESS>
      <STREET>2 Avocet Drive</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
      <ZIP>95054</ZIP>
    </ADDRESS>
    <PHONELIST>
      <VARCHAR2>415-555-1212</VARCHAR2>
    </PHONELIST>
  </CUSTOMER>
</ORDERDATE>10-APR-97</ORDERDATE>
```

```
<SHIPDATE>10-MAY-97 12.00.00.000000 AM</SHIPDATE>
<LINEITEMS_NTAB>
  <LINEITEM_TYP LineItemNo="1">
    <ITEM StockNo="1534">
      <PRICE>2234</PRICE>
      <TAXRATE>2</TAXRATE>
    </ITEM>
    <QUANTITY>12</QUANTITY>
    <DISCOUNT>0</DISCOUNT>
  </LINEITEM_TYP>
  <LINEITEM_TYP LineItemNo="2">
    <ITEM StockNo="1535">
      <PRICE>3456.23</PRICE>
      <TAXRATE>2</TAXRATE>
    </ITEM>
    <QUANTITY>10</QUANTITY>
    <DISCOUNT>10</DISCOUNT>
  </LINEITEM_TYP>
</LINEITEMS_NTAB>
<SHIPTOADDR/>
</PurchaseOrder>
<PurchaseOrder>
  <PONO>2001</PONO>
  <CUSTOMER>
    <CUSTNO>2</CUSTNO>
    <CUSTNAME>John Nike</CUSTNAME>
    <ADDRESS>
      <STREET>323 College Drive</STREET>
      <CITY>Edison</CITY>
      <STATE>NJ</STATE>
      <ZIP>08820</ZIP>
    </ADDRESS>
    <PHONELIST>
      <VARCHAR2>609-555-1212</VARCHAR2>
      <VARCHAR2>201-555-1212</VARCHAR2>
    </PHONELIST>
  </CUSTOMER>
  <ORDERDATE>20-APR-97</ORDERDATE>
  <SHIPDATE>20-MAY-97 12.00.00.000000 AM</SHIPDATE>
<LINEITEMS_NTAB>
  <LINEITEM_TYP LineItemNo="1">
    <ITEM StockNo="1004">
      <PRICE>6750</PRICE>
      <TAXRATE>2</TAXRATE>
    </ITEM>
```

```
<QUANTITY>1</QUANTITY>
<DISCOUNT>0</DISCOUNT>
</LINEITEM_TYP>
<LINEITEM_TYP LineItemNo="2">
  <ITEM StockNo="1011">
    <PRICE>4500.23</PRICE>
    <TAXRATE>2</TAXRATE>
  </ITEM>
  <QUANTITY>2</QUANTITY>
  <DISCOUNT>1</DISCOUNT>
</LINEITEM_TYP>
</LINEITEMS_NTAB>
<SHIPTOADDR>
  <STREET>55 Madison Ave</STREET>
  <CITY>Madison</CITY>
  <STATE>WI</STATE>
  <ZIP>53715</ZIP>
</SHIPTOADDR>
</PurchaseOrder>
</ROWSET>
```

Other Aggregation Methods

ROLLUP and CUBE

Oracle provides powerful functionality for OLAP operations such as CUBE and ROLLUP. `SYS_XMLAGG` function also works in these cases. You can, for example, create different XML documents based on the ROLLUP or CUBE operation.

WINDOWING Function

Oracle provides windowing functions that can be used to compute cumulative, moving, and centered aggregates. `SYS_XMLAGG` can also be used here to create documents based on rank and partition.

TABLE Functions

The previous sections talked about the new functions and operators that Oracle has introduced that can help query the XML instances using a XPath-like syntax. However, you also need to be able to explode the XML into simple relational or object-relational data so that you can insert that into tables or query using standard relational SQL statements. You can do this using a powerful mechanism called TABLE functions.

Table functions are new in Oracle. They can be used to model any arbitrary data (internal to the database or from an external source) as a collection of SQL rows. Table functions are executed pipelined and in parallel for improved performance. You can use Table functions to break XML into SQL rows. These can then be consumed by regular SQL queries and inserted into regular relational or object-relational tables.

With Oracle8i you could have a function returning a collection and use it in the FROM clause in the SELECT statement. However, the function would have to materialize the entire collection before it can be consumed by the outer query block.

With TABLE functions, these are both pipelined and parallel. Thus the function need not instantiate the whole collection in memory and instead pipe the results to the outer query block. The result is a faster response time and lower memory usage.

See Also: *Oracle9i Application Developer's Guide - Fundamentals*, Table Functions.

Using Table Functions with XML

With XML, you can use these TABLE functions, to break the XML into SQL rows that can be queried by regular SQL queries and put into regular relational or object relational tables. Since they are parallel and piped, the performance of such an operation is vastly improved.

You can define a function for instance that takes an XMLType or a CLOB and returns a well known collection type. The function, for example, can use the XML parser available with Oracle to perform SAX parsing and return the results, or use the `extract()` function to extract pieces of the XML document and return it.

Table Functions Example 1: Exploding the PO to Store in a Relational Table

Assuming that you have the purchase order document explained in earlier sections and you need to explode it to store it in relational table containing the purchase order details, you first create a type to describe the structure of the result,

```
create type poRow_type as object
(
  poname varchar2(20),
  postreet varchar2(20),
  pocity varchar2(20),
  postate char(2),
  pozip char(10)
);
```

```

/
create type poRow_list as TABLE of poRow_type;
/

```

Now, you can either create an ODCI implementation type to implement the TABLE interface, or use native PL/SQL.

See Also: *Oracle9i Application Developer's Guide - Fundamentals*, Table Functions, for details on what the interface definitions are and an example of the body,...

Assuming that you have created the body of the implementation type in PL/SQL, by creating the function itself, you can define the TABLE function as follows:

```

create function poExplode_func (arg IN sys.XMLType) return poRow_list
pipelined is
  out_rec poRow_type;
  poxml sys.XMLType;
  i binary_integer := 1;
  argnew sys.XMLType := arg;
begin

  loop

    -- extract the i'th purchase order!
    poxml := argnew.extract('//PO['||i||']');
    exit when poxml is null;

    -- extract the required attributes...!!!
    out_rec.poname := poxml.extract('/PONAME/text()').getStringVal();
    out_rec.postreet := poxml.extract('/POADDR/STREET/text()').getStringVal();
    out_rec.pocity := poxml.extract('/POADDR/CITY/text()').getStringVal();
    out_rec.postate := poxml.extract('/POADDR/STATE/text()').getStringVal();
    out_rec.pozip := poxml.extract('/POADDR/ZIP/text()').getStringVal();
    PIPE ROW(out_rec);

    i := i+1;

  end loop;
  return;
end;
/

```

You can use this function in the FROM list, and the interfaces defined in the Imp_t would be automatically called to get the values in a pipelined fashion as follows:


```

select *
  from TABLE( CAST(
    poExplode_func(
      sys.XMLType.createXML(
        '<?xml version="1.0"?>
        <POLIST>
          <PO>
            <PONAME>Po_1</PONAME>
            <POADDR>
              <STREET>100 Main Street</STREET>
              <CITY>Sunnyvale</CITY>
              <STATE>CA</STATE>
              <ZIP>94086</ZIP>
            </POADDR>
          </PO>
          <PO>
            <PONAME>Po_2</PONAME>
            <POADDR>
              <STREET>200 First Street</STREET>
              <CITY>Oaksdale</CITY>
              <STATE>CA</STATE>
              <ZIP>95043</ZIP>
            </POADDR>
          </PO>
        </POLIST>' )
    ) AS poRow_list));

```

Note: IN the foregoing example, XMLType static constructor was used to construct the XML document. You can also use bind variables or select list subqueries to get the value.

The SQL statement returns the following values:

PONAME	POSTREET	POCITY	POSTATE	POZIP
Po_1	100 Main Street	Sunnyvale	CA	94086
Po_2	200 First Street	Oaksdale	CA	95043

which can then be inserted into relational tables or queried with SQL.

Frequently Asked Questions (FAQs): XMLType

Is Replication and Materialized Views (MV) Supported by XMLType?

Question Are there any issues regarding using XMLType with replication and materialized views?

Answer Replication treats XMLType as another user-defined type. It should work as other user-defined types. dbms_defer_query and user-defined conflict resolution routines do not support XMLType until a forthcoming release. So, no, replication and MV are not fully supported by XMLType in this release.

How Can I Update an XML Tag in a Database Record?

Question Can I update an XML tag within a record of a database table? Also, can I create indexes based on XML tags?

Answer In this release Oracle offers XMLType with a CLOB storage. Updatability is at the level of a single document, so you must replace the whole document.

XMLType can be indexed using Oracle Text (*interMedia*). This kind of index is best for locating documents that match certain XML search criteria, but if your applications are going to want to select out data like the “quantity” and the “price” as numerical values to operate on (for example, by some graphing or data warehousing software), then storing data-oriented XML as real tables and columns will give the best fit. You can also use functional indexes to speed up certain well-known XPath expressions.

See Also:

- [Chapter 8, "Searching XML Data with Oracle Text"](#)
- *Oracle9i Text Developer's Guide*
- *Oracle9i Text Reference*

The book “Building Oracle XML Applications”, by Steve Muench (O’Reilly), covers Oracle8i with examples.

Does XMLType Support the Enforcing of Business Rules Such as Attribute Constraints?

Question Does the XML datatype in Oracle, provide the ability to enforce business rules or constraints as an attribute of an element within the data? For example:

```
<Address>
  <Street type=string> </Street>
  <City type=string> </City>
  <State type=string size=2> </State>
  <Zip type=number size=5> </Zip>
</Address>
```

Answer With Oracle, you can use the trigger mechanism to enforce any constraints. Oracle currently only supports storage as a CLOB and no constraint checks are inherently available. With forthcoming releases we may offer XML schema compliance and you would be able to create columns conforming to schemas, and so on. This would enable you to have any constraint that the schema enforces.

How Do I Create XML Documents with the Appropriate Encoding for Japanese?

Question I need to use `SYS_XMLGEN()` to create documents in Japanese, on Oracle. The database character set is `EUC_JP`, but creating XML with `SYS_XMLGEN()`, `SYS_XMLAGG()` and `DBMS_XMLGEN()` produces documents with the XML declaration:

```
<?xml version="1.0"?>
```

Can I create XML documents with the appropriate declaration, such as:

```
<?xml version="1.0" encoding="EUC_JP"?>
```

Answer In Oracle, you cannot generate XML output with an encoding tag. If you have data in binary datatype, then the conversion will fail if it is not convertible to the database character set.

Note that the encoding declaration does not constitute a part of the document per-se. It is an identifier to help processors identify its encoding.

What is the Best Way to Store XML in a Database?

Question What is the best way to store XML in a database? Can I store XML data in an NCLOB? Or is it preferable to store it in a BLOB and manipulate it within the application only?

Answer The best way to store XML in Oracle is to use the XMLType. Oracle currently allows only CLOB storage natively, but in forthcoming releases Oracle may allow native storage in NCLOBs, BLOBs, etc.

For Oracle Release 10g (9.0.4), you can store the XML as:

- XMLType - the preferred way. This helps the server to know that the data is XML. You can do XML based querying and indexing.
- BLOBs - This helps you store the XML document intact in the same encoding as the original document. The burden is on the user to deal with all the encoding issues.
- NCLOBs - Storing XML as NCLOBs will allow you to SQL operations on the data, as well as do Context searches. Context search is supported only if the database charset is a proper super-set or convertible set of the NCHAR, that is, the NCLOB must be convertible to the database char set without loss of data.

Database Uri-references

This chapter contains the following sections:

- [Uri-reference \(Uri-ref\) Concepts](#)
- [New Datatypes Store Uri-references](#)
- [DBUri-refs, Intra-Databases References](#)
- [Using Uri-ref Types \(URITypes\)](#)
- [UriFactory Package](#)
- [Why Use Different Uri-refs?](#)
- [SYS_DBURIGEN\(\) SQL Function](#)
- [Accessing DBUri-refs From Your Browser Using Servlets](#)

Uri-reference (Uri-ref) Concepts

What is a Uri-ref?

Uri-Reference (Uri-ref) is a generalization of the URL concept. In this release, URI can reference any document, including HTML and XML. It also provides pointer semantics into the document. Uri-ref consists of two parts:

- **URL part**, according to the regular URL specification
- **Fragment part**, that identifies a fragment within that document.

It is in a language specific to the type of the document in question. The fragment part is that part after the "#" in the following examples. The fragment part is not supported in this release.

URL Path Created From an XML Document View

[Figure 6-2](#) shows a view of the XML data stored in relational table, Emp, in the database, and the columns of data mapped to elements in the XML document. This mapping is also referred to as an "XML visualization". The resulting URL path can then be created simply from the XML document view.

Typical Uri-ref's look like the following:

- **For HTML:** `http://www.url.com/document1#A`
where A is an anchor inside the document.
- **For XML:** `http://www.xml.com/xml_doc#//po/cust/custname`
where:
 - The portion before the '#' identifies the location of the document itself
 - The portion after the '#' identifies a fragment within that document. This portion is defined by W3C's XPointer specification.

Uri-ref can Use Different Protocols to Retrieve Data

Oracle, has introduced new datatypes in the database to store and retrieve uri-ref objects. See ["New Datatypes Store Uri-references"](#) in the following section. Uri-ref, can in turn, use different protocols, such as the HTTP, to retrieve data.

Oracle has also introduced a new concept called **DBUri-refs**. These are references into columns and rows of tables and views inside the database itself.

- HTTP URL references objects that are global

- DBUri-ref references *local* objects

Using this DBUri-ref mechanism you can access any row or column data in any table or view in the database. In effect it provides a *intra-database URL for any data stored in the database*. See "[DBUri-refs, Intra-Databases References](#)" on page 6-4.

Advantages of Using DBUri-ref

DBUri-ref advantages include the following:

- **Improved Performance by Bypassing the Web Server.** Uri-ref is one way to locally reference stylesheets. For example, DBMS-Metadata uses DbUri-ref to reference many stylesheets. If you already have a URL in your XML document and you have to replace this with a reference to the database, you can either:

- Use a servlet
- Use an input mechanism, such as the DBUri-ref, to bring back the results

Using DBUri-ref has performance benefits as you interact directly with the database rather than through a web server.

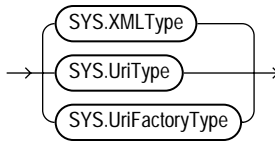
- **Who Needs SQL?** You do not need to know SQL to access an XML document stored in the database. DBUri-ref allows you to access an XML document from the database using SQL semantics but without your need to use SQL. In other words, a non-SQL programmer can now easily access XML documents stored in a database.

New Datatypes Store Uri-references

Oracle has introduced the following new datatypes to store the uri-references:

- **UriType.** An abstract object type that can store instances of HttpUriType or DBUriType.
- **DBUriType.** Can obtain data pointed to by the DBUri-ref.
- **HttpUriType.** Implements the HTTP protocol for accessing remote pages.
- **UriFactoryType.**

See Also: [Figure 6-3, "UriFactory to Generate UriType Instances"](#).

Figure 6–1 New UriTypes

These datatypes are object types with member functions that can be used to access objects or pages pointed to by the objects. Thus by using the UriType, you can do the following:

- Create columns that can point to data inside or outside the database
- Query the database columns using abstract functions provided by UriType

Benefits of Using UriTypes

Oracle already supports UTL_HTTP and java.net.URL in PL/SQL and Java respectively, to fetch URL references. The advantages of defining this new UriType datatype in SQL are as follows:

- **Better Typing Leads to More Efficient Indexing, Navigation, and Querying**
Uri-ref also makes the database aware of a new URL type that can be stored in columns of tables or views. Better typing of a column can lead to newer url-aware indexing schemes and better navigation and query capabilities through the URL.
- **Improved Mapping of XML Documents to Columns.** Uri-ref support is needed when exploding XML documents into object-relational columns, so that the Uri-ref specified in documents can map to a URL column in the database.

See Also: ["Using Uri-ref Types \(URITypes\)"](#) on page 6-14.

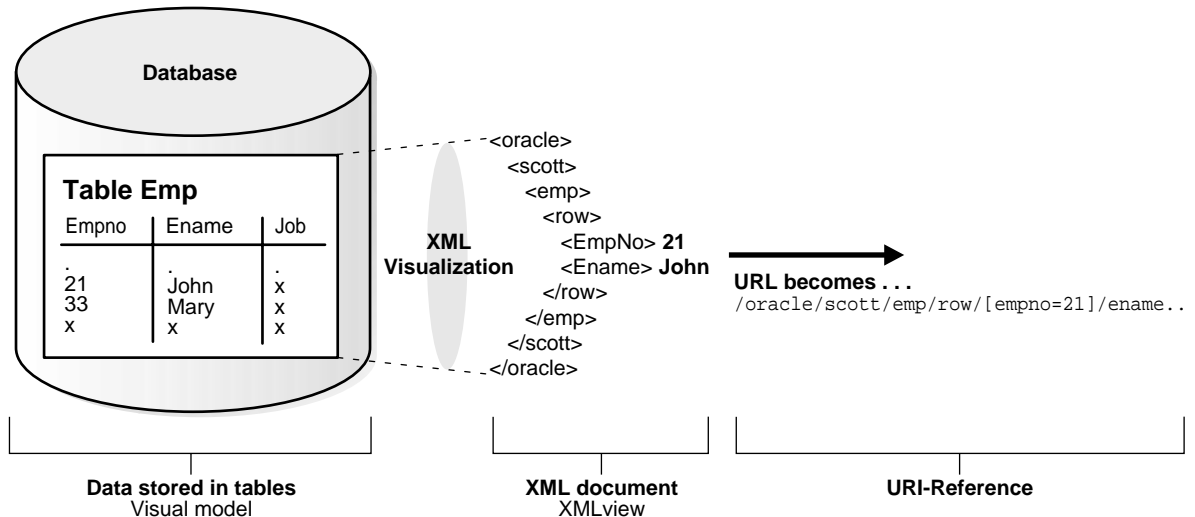
DBUri-refs, Intra-Databases References

DBUri-Ref, a database relative to URI, is a special case of the Uri-ref mechanism, where ref is guaranteed to work inside the context of a database and session. This ref is not a global ref like the HTTP URL, instead it is local ref (URL) within the database.

You can also access objects pointed to by this URL globally, by “appending” this DBUri-ref to an HTTP URL path that identifies the servlet that can handle

DBUri-ref. This is explained later under "[Accessing DBUri-refs From Your Browser Using Servlets](#)" on page 6-25.

Figure 6-2 DBUri-ref Explained



Formulating the DBUri

The URL syntax is obtained by specifying a XPath-like syntax over a virtual XML visualization of the database. See also [Figure 6-2, "DBUri-ref Explained"](#).

The "visual model" is a hierarchical view of what a current connected user would see in terms of SQL schemas, tables, rows and columns.

The "XML view" contains a root element that maps to the database. The root XML element contains child elements which are the schemas on which the user has some privileges on any object. The schema elements contain tables and views which the user can see. For example, the user *scott* can see the following virtual document.

```
<?xml version='1.0'?>
<oradb SID="ORCL">
  <PUBLIC>
    <ALL_TABLES>
      ..
    </ALL_TABLES>
  <EMP>
```

```
        <!-- Emp table -->
    </EMP>
</PUBLIC>
<SCOTT>
  <ALL_TABLES>
    . . . .
  </ALL_TABLES>
</EMP>
  <ROW>
    <EMPNO>1001</EMPNO>
    <ENAME>John</ENAME>
    <EMP_SALARY>20000</EMP_SALARY>
  </ROW>
  <ROW>
    <EMPNO>2001</EMPNO>
    <ENAME xsi:null="true"/>
    <EMP_SALARY xsi:null="true"/>
  </ROW>
</EMP>
<DEPT>
  <ROW>
    <DEPTNO>200</DEPTNO>
    <DNAME>Sports</DNAME>
  </ROW>
</DEPT>
</SCOTT>
<JONES>
  <CUSTOMER_OBJ_TAB>
    <ROW>
      <NAME>xxx</NAME>
      <ADDRESS>
        <STATE>CA</STATE>
        <ZIP>94065</ZIP>
      </ADDRESS>
    </ROW>
  </CUSTOMER_OBJ_TAB>
</JONES>
</database>
```

Remember, that this is a virtual XML document based on the privileges that you have at the time of access.

You can make the following observations from the foregoing example:

- The *scott* user can see the *scott* schema and *jones* schema. These are schemas on which the user has some table or views that he can read.
- Table *emp* shows up as EMP with row element tags. This is the default mapping for all tables. The same for dept and the customer_obj_tab table under the JONES schema.
- Null elements are shown as being absent. This will change with later releases to conform to the XMLSchema specification which specifies a special null attribute to indicate nullness.
- There is also a PUBLIC element under which are tables and views accessible without schema qualification. For example, a select query such as:

```
select * from emp;
```

when queried by the user *scott*, will match the table emp under the *scott* schema and if not found, would try to match it with a public synonym that is available. In the same way, the PUBLIC element contains all the tables and views that are either visible to the user through his/her schema and all the tables that are visible through the PUBLIC synonym.

The DB-Uri Specification

With the database being visualized as an XML tree, you can perform XPath traversals to any part of the virtual document. This translates to any row-column intersection of the database tables or views. By specifying an XPath over the visualization model, you can create *references* to any piece of data in the database.

DbUri is specified in a *simplified* XPath format. For this release Oracle does not support the full flavor of XPath or Xpointer for DBUri- ref. The following sections discuss the structure of these DBUri.

Note: When exposing the DBUri through a global HTTP URL, you may have to "escape" certain characters such as ']',',',.... in the XPath syntax. You can use the `getExternalUrl()` functions in the types to get an escaped version of the URL.

As stated above, you can now create DBUri's to any piece of data. You can use the following units of reference:

- A *scalar* or *object* or *collection instance* in a column
- An attribute of an object type

Note: Oracle does not currently support references within a scalar, XMLType or LOB data value.

With DBUri's, you can also create globally reference able URLs. This is explained in a later section, "[Accessing DBUri-refs From Your Browser Using Servlets](#)" on page 6-25.

DBUri Syntax Guidelines

There are restrictions on the kind of XPath queries that can be used to specify a reference. In general, DBUri-ref's must adhere to the following syntax guidelines:

- Include the user schema name or PUBLIC to resolve the table name without a specific schema.
- Include a table or view name.
- Include the ROW tag for identifying the ROW element.
- Identify the column or object attribute that you wish to extract.
- Include predicates at any level in the path other than the schema and table elements.
- Indicate predicates not on the selection path in the ROW element. For example, if you wanted to specify the predicate, pono = 100, but the selection path is /scott/purchase_obj_tab[.]/ROW/line_item_list, then you must include the pono predicate along with the ROW node as:

```
/scott/purchase_obj_tab/ROW[spono=100]//line_item_list
```

- DBUri-ref *must* identify exactly a single data value which may be an object type or collection. The data value can be an entire row in which case ROW node must be used to indicate that. The uri-ref can also point to an entire table.

Using Predicate (XPath) Expressions in DBUri

The predicate expressions can use the following XPath expressions:

- Boolean operators - AND, OR and NOT
- Relational operators - <, >, <=, !=, >=, =, mod, div, * (multiply)

Note:

- No XPath axes other than the child axes are supported. Thus the wild card (*), descendant (/ /), and other operations are not valid.
 - No XPath functions other than text() are supported. text() is also valid only on a scalar node. Thus you cannot apply the text() node say at the ROW level or at the table level
-
-

The predicates can be defined at any element other than the schema and table elements. If you have object columns, then you can search on the attribute values as well. For example, if address was a column in the emp table, which contains say the state, city, street and zipcode attributes, then the following dburi-ref is valid:

```
/SCOTT/EMP/ROW[ADDRESS/STATE='CA' OR ADDRESS/STATE='OR']/ADDRESS[CITY='Portland'
OR ./ZIPCODE=94404]/CITY
```

This dburi-ref identifies the city attribute of the address column in the emp table whose state is either California or Oregon or the city name is Portland or the zipcode is 94404.

Some Common DBUri-ref Scenarios

The DBUri-ref can identify various objects, such as the table, a particular row, a particular column in a row, or a particular attribute of an object column. Here are some common DBUri-ref scenarios:-

1. **Identifying the whole table.** This returns an XML document that retrieves the whole table. The enclosing tag is the name of the table. The row values are enclosed inside a "ROW" element, as follows:

Use the following syntax:

```
/<schemaname>/<tablename>
```

For example:

```
/SCOTT/EMP
```

returns the following XML document,:

```
<?xml version="1.0"?>
<EMP>
```

```
<ROW>
  <EMPNO>7369</EMPNO>
  <ENAME>Smith</ENAME>
  ... <!-- other columns -->
</ROW>
<!-- other rows -->
</EMP>
```

- 2. Identifying a particular row of the table.** This identifies a particular ROW element in a table. The result is an XML document that contains the ROW element with its columns as child elements:

Use the following syntax:

```
/<schemaname>/<tablename>/ROW[<predicate_expression>]
```

For example:

```
/SCOTT/EMP/ROW[EMPNO=7369]
```

returns the following XML document:

```
<?xml version="1.0"?>
<ROW>
  <EMPNO>7369</EMPNO>
  <ENAME>SMITH</ENAME>
  <JOB>CLERK</JOB>
  <!-- other columns -->
</ROW>
```

Note: Here, the predicate expression must identify a unique row.

- 3. Identifying a target column.** In this case a target column or an attribute of a column is identified and retrieved as XML.

Note: You cannot traverse into nested table or VARRAY columns.

Use the following syntax:

```
/<schemaname>/<tablename>/ROW[<predicate_expression>]/<columnname>
/<schemaname>/<tablename>/ROW[<predicate_expression>]/<columnname>/
<attribute>*
```

Example 1:

```
/SCOTT/EMP/ROW[EMPNO=7369 and DEPTNO =20]/ENAME
```

retrieves the ename column in the emp table, where empno is 7369, and department number is 20, as follows:

```
<?xml version="1.0"?>
<ENAME>SMITH</ENAME>
```

Example 2:

```
/SCOTT/EMP/ROW[EMPNO=7369]/ADDRESS/STATE
```

retrieves the state attribute inside an address object column for the employee whose empno is 7369, as follows:

```
<?xml version="1.0"?>
<STATE>CA</STATE>
```

4. **Retrieving the text value of a column.** In many cases, it can be useful to retrieve only the text values of a column and not the enclosing tags. For example, if the XSL stylesheets are stored in a CLOB column, you can retrieve such XML documents without having an enclosing column name tag put around them. In these cases, the `text()` function helps identify that only the text value of the node is to be retrieved.

Use the following syntax:

```
/<schemaname>/<tablename>/ROW[<predicate expression>]/<columnname>/text()
```

For example:

```
/SCOTT/EMP/ROW[EMPNO=7369]/ENAME/text()
```

retrieves the text value of the employee name, without the XML tags, for employee with empno = 7369. This returns a text document, not an XML document, with value "SMITH".

Note: The XPath alone does not constitute a valid URI. Oracle calls it a DBUri since it behaves like a URI within the database, but it can be translated in to a globally valid Uri-Ref.

Note: The DBUri is case-sensitive. So to specify scott.emp, you use SCOTT/EMP since the actual table names are stored capitalized in the Oracle dictionary. To create a table name or column name in small letters in the database use the " " to enclose the names.

How DBUri's Differ from Object References

DBUri-ref has *column* and *attribute* level access and is loosely typed. Oracle8 object features provide object references which are references to *row objects* in the system. DBUri-ref is inherently a superset of this reference mechanism.

DBUri-ref can not only identify a particular row, but can also provide *access to a column or an object attribute* of the row. However, it is loosely typed unlike the object reference. The result of the Uri-ref traversal can be an object in the system.

DBUri-ref Applies to a Database and Session

An important aspect of DBUri-ref is that it is scoped to a database and session. Since DBUri-ref itself does not carry any session specific information, it is assumed that you are connected to the database in a particular session context and are resolving the Uri-ref in that context. This is similar to the object reference mechanism, where the dereferencing of an object reference requires that you have privileges to read the referenced object.

Note: The same DBUri-ref may give different results based on the session context used, particularly if the PUBLIC path is used.

For example, /PUBLIC/FOO_TAB can resolve to SCOTT.FOO_TAB when connected as scott, and resolve as JONES.FOO_TAB when connected as JONES.

Where Can DBUri-ref be Used?

Uri-ref can be used in a number of scenarios, including the following:

For Storing URLs to Related Documents

In the case of a travel story web site where you store travel stories in a table, you may have to create links to related stories. DBUri's can help, since you can create an intra-database link to the related story.

For Storing Stylesheets in the Database

Applications can use XSL stylesheets to convert XML into other formats. This data is transformed into XML. The XSL stylesheets used are stored in CLOBs. The application can use DBUri references in the following manner:

- To access the XSL stylesheets stored in the database for use during parsing
- To have references to related XSL stylesheets, such as, import/include,... within the XSL stylesheet itself.

Note: DBUri-ref does not provide the following support:

- As a general purpose XPointer mechanism to XML data.
 - It is not a replacement for database object *references*. The syntax and semantics of *references* differ from those of Uri-ref type.
 - It does not enforce or create any new security models or restrictions. Instead, it relies on the underlying security architecture to enforce privileges.
-
-

Using Uri-ref Types (URITypes)

This section describes how to use Uri-ref to store pointers to documents and how to access such Uri-refs in the database.

Storing Pointers to Documents with UriType

As explained earlier, UriType is an abstract type that can store instances of its subtypes in a column. This type contains a single VARCHAR2 attribute containing the Uri-ref string and has functions for traversing the reference and extracting the data.

You can create columns of UriType and store Uri-references in the database. Oracle provides standard classes for HTTP and DBUri traversals. You can navigate the URI using a rich set of navigational functions.

Table 6–1 lists some useful UriType methods.

Note: You can plug-in any new protocol using the inheritance mechanism. Oracle provides the HttpUriType and DBUriType types for handling HTTP protocol and for deciphering DBUri references. You can for instance, implement a subtype of the UriType, to handle say, the *gopher* protocol.

Table 6–1 URIType Methods

URIType Method	Description
getClob	This returns the value pointed to by the URL as a character lob value. The character encoding will be that of the database character set.
getUrl	Returns the url that is stored in the UriType. Do not use the attribute "url" directly. Use this function instead. This can be overridden by subtypes to give you the correct url. For instance, the HttpUriType, stores only the URL and not the "http://" prefix. So the getUrl() actually appends the prefix and returns the value.
getExternalUrl	Similar to the latter (getUrl), except that it calls the escaping mechanism to escape the characters in the URL as to conform to the URL specification. (For example spaces are converted to the escaped value %20)

URIType Examples

URIType Example 1: Creating URL References to a List of Purchase Orders

You can create a list of all purchase orders with URL references to these purchase orders as follows:

```

CREATE TABLE uri_tab
(
  poUrl SYS.UriType, -- Note that we have created abstract type columns
-- if we knew what kind of uri's we are going to store, we can actually
-- create the appropriate types.
  poName VARCHAR2
);

-- insert an absolute url into SYS.UriType..!
-- the factory will create the correct instance (in this case a FtpUriType
INSERT INTO uri_tab VALUES
  (sys.UriFactory.getUri('http://www.oracle.com/cust/po'), 'AbsPo');

-- insert a URL by directly calling the SYS.HttpUriType constructor.
-- Note this is strongly discouraged. Note the absence of the
-- http:// prefix when creating SYS.HttpUriType instance through the default
-- constructor.
INSERT INTO uri_tab VALUES (sys.HttpUriType('proxy.us.oracle.com'), 'RelPo');

-- Now extract all the purchase orders
SELECT e.poUrl.getClob(), poName FROM uri_tab e;

-- In PL/SQL
declare
  a SYS.UriType;
begin

  -- absolute URL
  SELECT poUrl into a from uri_Tab WHERE poName like 'AbsPo%';
  printDataOut(a.getClob());

  SELECT poUrl into a from uri_Tab WHERE poName like 'RelPo%';
  -- here u need to supply a prefix before u can get at the data..!
  printDataOut(a.getClob());
end;
/

```

See: ["UriFactory Package"](#) on page 6-17 for a description on how to use URIFactory.

URIType Example 2: Using the Substitution Mechanism

You can create columns of the UriType directly, and insert both HttpUriTypes and DBUriTypes into that column. You can also query the column without knowing where the referenced document lies.

For example, from the first example, you can insert DBUri-ref references as well into the *uri_tab* table as follows:

```
INSERT INTO uri_tab VALUES
  (UriFactory.getUrl(
    '/SCOTT/PURCHASE_ORDER_TAB/ROW[PONO=1000]'), 'ScottPo');
```

This insert, assumes that there is a purchase order table in the SCOTT schema. Now, the url column in the table contains values that are pointing through HTTP to documents globally as well as pointing to virtual documents inside the database.

A select on the column using the `getClob()` method, would retrieve the results as a CLOB irrespective of where the document resides:

```
select e.poURL.getClob() from uri_tab e;
```

would retrieve values from the global HTTP address stored in the first row as well as the local DBUri reference.

Using HttpUriType and DBUriType

HttpUriType and DBUriType are sub types of UriType and implement the functions for HTTP and DBUri-ref references respectively.

Note: HttpUriType cannot store relative HTTP references, in this release.

DBUriType Examples

DBUriType Example 1: Creating DBUri-ref References

The following example creates a table with a column of type DBUriType and assigns a value to it.

```
CREATE TABLE DBURITab(DBUri DBUriType, dbDocName VARCHAR2(2000));
```

```
-- insert values into it..!  
INSERT INTO DBUriTab VALUES  
    (sys.UriFactory.createUri('/ORADB/SCOTT/EMP/ROW[EMPNO=7369]'), 'emp1');  
  
INSERT INTO DBUriTab VALUES  
    (sys.DBUriType('/SCOTT/EMP/ROW[EMPNO=7369]/'), 'emp2');  
  
-- access the references  
SELECT e.DBUri.getCLOB() from dual;
```

UriFactory Package

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. See [Figure 6-3](#).

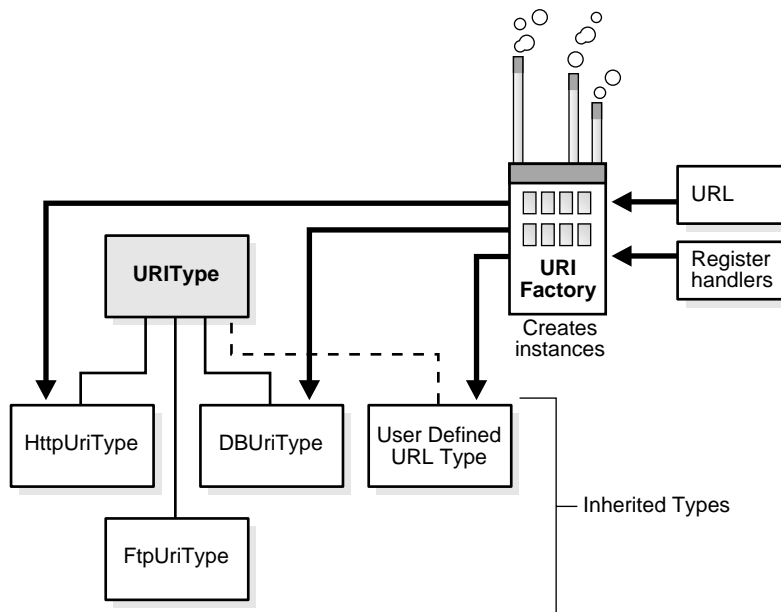
The factory method can take in strings representing the various URLs and return the appropriate subtype instance. For example:

- If the prefix starts with `http://`, it creates a `SYS.HttpUriType` and returns a reference to that instance, after stripping out the `http://` prefix.
- If the string starts with a `"/oradb/"` prefix or does not match any of the well known prefixes (such as `http://...`), then it creates a `SYS.DBUriType` instance and returns that.

Registering New UriType Subtypes

The UriFactory package also provides the ability to register new subtypes of the UriType to handle various other protocols not currently supported by Oracle. For example, you 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.

Figure 6–3 UriFactory to Generate UriType Instances



UriFactory Example: Registering the ecom Protocol

To register the new protocol `ecom://`. You need to do the following:

- Define a type to handle the protocol
- Register it with the UriFactory package, as follows:

```

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 SYS.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'); -- a Http uri type instance

SYS.DBUriType('/SCOTT/EMPLOYEE/ROW[ENAME="Jack"],null); -- a SYS.DBUriType

EComUriType('company1/company2=22/comp'); -- a EComUriType instance

```

Why Use Different Uri-refs?

As explained in earlier sections, Uri-ref is an abstract class with various derivations which implement different protocols. The advantage of separation of the various derivations is two fold.

- If you choose a subtype for representing a column, it provides an implicit constraint on the column to contain only instances of that protocol type. This might be useful for implementing specialized indexes on that column for specific protocols. For example for the DbUri-ref you can implement some specialized indexes that can directly go and fetch the data from the disk blocks rather than executing SQL queries.

- The second reason is that, you can have different constraints on the columns based on the type involved. For instance for the HTTP case, you could potentially define proxy and firewall constraints on the column so that any access through the HTTP would use the proxy server.

The separation of the implementation classes from the abstract Uri-ref class provides:

- Better modelling
- Better extension capabilities.

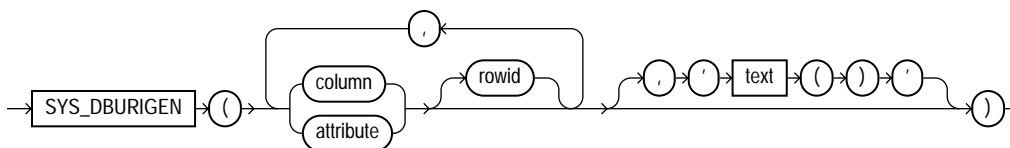
You can now implement your own protocol and actually make the database treat that as an Uri-ref for purposes of navigation, indexing, and so on.

SYS_DBURIGEN() SQL Function

DBUri reference can be created by specifying the path expression to the constructor or the UriFactory methods. However, you also need methods to generate these DBUri references dynamically given target columns. For this purpose a new SQL function, called `SYS_DBURIGEN()`, has been introduced.

Figure 6–4 shows the `SYS_DBURIGEN()` syntax.

Figure 6–4 *SYS_DBURIGEN() Syntax*



The following example uses the `SYS_DBURIGEN()` function to generate a URL of datatype `DBUriType` to the email column of the row in the sample table `hr.employees` where the `employee_id = 206`:

```
SELECT SYS_DBURIGEN(employee_id, email)
       FROM employees
       WHERE employee_id = 206;

SYS_DBURIGEN(EMPLOYEE_ID,EMAIL)(URL, SPARE)
-----
DBURITYPE('/PUBLIC/EMPLOYEES/ROW[EMPLOYEE_ID = "206"]/EMAIL', NULL)
```


`SYS_DBURIGEN()` function takes as its argument one or more columns or attributes, and optionally a rowid, and generates a URL of datatype `DBUriType` to a particular column or row object. You can then use the URL to retrieve an XML document from the database. The function takes an additional parameter to indicate if the text value of the node is needed.

All columns or attributes referenced must reside in the same table. They must perform the function of a primary key. That is, they need not actually match the primary keys of the table, but they must reference a unique value. If you specify multiple columns, all but the final column identify the row in the database, and the last column specified identifies the column within the row.

By default the URL points to a formatted XML document. If you want the URL to point only the text of the document, specify the optional `'text()'`. (In this XML context, the lowercase `'text'` is a keyword, not a syntactic placeholder.)

If the table or view containing the columns or attributes does not have a schema specified in the context of the query, Oracle interprets the table or view name as a public synonym.

The column or attribute passed to the `SYS_DBURIGEN()` function must obey the following rules:

- **Unique mapping:** The column or UDT attribute must be uniquely mappable back to the table or view from which it comes. This means that virtual columns are not allowed. The only exception is the `VALUE` and `REF` operators. The column may come from a `TABLE()` subquery or an online view provided that the online view does not rename the columns.
- **Key columns:** *Either the rowid or a set of key columns must be specified.* The list of key columns need not match the list of keys in the table so long as the columns can uniquely identify a particular row in the result.
- **Same table:** All columns referenced in the `SYS_DBURIGEN()` function **MUST** come from the same table or view.
- **PUBLIC element:** If the table or view pointed by the rowid or key columns does not have a database schema specified, then the `PUBLIC` keyword would be used instead of the schema. This has the effect of using the current binding (that is, table or synonym or view) to the table name when the `DBUri` is accessed.
- **TEXT function:** `DBUri`, by default, retrieves an XML document containing the result. To retrieve only the text value, use the `'text()'` keyword as the final argument to the function.

For example:

```
select SYS_DBURIGEN(empno,ename,'text()') from scott.emp,
```

generates a URL of the form:

```
/SCOTT/EMP/ROW[EMPNO=7369]/ENAME/text()
```

- **Single column argument:** If there is a single column argument, then the column is used both as the key column to identify the row and as the referenced column. For example:

```
select SYS_DBURIGEN(empno) from emp;
```

would use the empno both as the key column and the referenced column, that is, it would generate a URL of the form:

```
/SCOTT/EMP/ROW[EMPNO=7369]/EMPNO,
```

for the row which has empno = 7369.

SYS_DBURIGEN Example 1: Inserting Database References

```
CREATE TABLE doc_list_tab(docno number primary key, doc_ref SYS.DBUriType);

-- inserts /SCOTT/EMP/ROW[rowid='xxx']/EMPNO
INSERT INTO doc_list_tab(1001,
    (select SYS_DBURIGEN(rowid,empno) from emp where empno = 100));

-- insert a Uri-ref to point to the empname column of emp!
INSERT INTO doc_list_tab
    select empno, SYS_DBURIGEN(empno, ename) from emp));

-- result of the DBURIGEN looks like, /SCOTT/EMP/ROW[EMPNO=7369]/ENAME
```

SYS_DBURIGEN Example 2: Returning Partial Results

When selecting the results of a column such as a lob column, you might want to retrieve only a portion of the result and create a URL to the column instead. For example, consider the case of a travel story web site. If you have a table which stores all the travel stories and the user queries over the table to find all relevant stories according to his search criterion, then you do not want to list the entire story in the result page. You, instead show the first 100 characters or the gist of the story and then return a URL to the actual story instead.

This can be done as follows:

Assume that the travel story table is defined as follows:

```
create table travel_story
(
  story_name varchar2(100),
  story clob
);
```

```
-- insert some value..!
```

```
insert into travel_story values ('Egypt','This is my story of how I spent my
time in Egypt, with the pyramids in full view from my hotel room');
```

Now, you create a function that returns only the first 20 characters from the story,

```
create function charfunc(clobval IN clob ) return varchar2 is
  res varchar2(20);
  amount number := 20;
begin
  dbms_lob.read(clobval,amount,1,res);
  return res;
end;
/
```

Now, you create a view which selects out only the first 100 characters from the story and then returns a DBUri reference to the story column.

```
create view travel_view as select story_name, charfunc(story) short_story,
  SYS_DBURIGEN(story_name,story,'text()') story_link
from travel_story;
```

Now, a select from the view returns the following:

```
select * from travel_view;
```

```
STORY_NAME      SHORT_STORY      STORY_LINK
-----
```

```
Egypt          This is my story of h
```

```
SYS.DBUriType('/PUBLIC/TRAVEL_STORY/ROW[SHORT_STORY='Egypt']/STORY/text()')
```

SYS_DBURIGEN Example 3: RETURNING Uri-refs

You can use `SYS_DBURIGEN` in the `RETURNING` clause of DML statements. This is useful to retrieve the URL to an object inserted. For example, consider table, `clob_tab`:

```
CREATE TABLE clob_tab ( docid number, doc clob);
```

If you insert a document, you may need to retrieve a URL to that document and store it in another table, `uri_tab`. This would be useful for auditing or other purposes.

```
CREATE TABLE uri_tab (docs sys.DBUriType);
```

You can do that as part of the insertion into `clob_tab`, using the `RETURNING` clause. You can use the `EXECUTE IMMEDIATE` syntax to execute the `SYS_DBURI` function inside PL/SQL as follows:

```
declare
  ret sys.dburitype;
begin
  -- exucute the insert and get the url
  EXECUTE IMMEDIATE
  'insert into clob_tab values (1, ''TEMP CLOB TEST'')
  RETURNING SYS_DBURIGEN(docid, doc, ''text()'') INTO :1 '
  RETURNING INTO ret;
  -- insert the url into uri_tab
  insert into uri_tab values (ret);
end;
/
```

The URL created would be of the form:

```
/SCOTT/CLOB_TAB/ROW[DOCID="xxx"]/DOC/text()
```

Note: The `text()` keyword is appended to the end, indicating that you want the URL to return just the CLOB value and not an XML document enclosing the CLOB text.

Accessing DBUri-refs From Your Browser Using Servlets

DBUri reference, is a database reference. It can be extended to be accessible from your browser or any other web server in the following ways:

- By writing a servlet that runs on the Oracle Servlet Engine or the Apache JServ module which can execute the DBUri-ref URL and return the values
- Through the default servlet provided to perform the same task. This servlet can only be run on the OSE as it uses JNI (Java Native Interface) to talk to the DBUri resolution code linked in to the server.

oracle.xml.dburi.OraDbUriServlet() Servlet Mechanism

For the above methods, a servlet, `OraDbUriServlet()` class, runs in Oracle servlet engine. This servlet takes in a path expression following the servlet name as the DbUri reference and outputs the document pointed to by the DBUri to the output stream. It can do either of the following:

- Generate the MIME type of the document automatically. In this release the only values supported are "text/xml" and "text/plain". In the case the DBUri ends in a `text()` function, then the "text/plain" mime type is used, else an XML document is generated with the mime type of "text/xml".
- Override the mime value using the *contenttype* argument to the servlet.

For example to retrieve the empno column of employee table, you can write a URL such as one of the following:

- `http://machine.oracle.com:8080/oradb/SCOTT/EMP/ROW[EMPNO=7369]/ENAME/text()` -- Generates a contenttype of text/plain
- `http://machine.oracle.com:8080/oradb/SCOTT/EMP/ROW[EMPNO=7369]/ENAME` -- Generates a contenttype of text/xml

where the machine machine.oracle.com is running the OSE, with a web service at port 8080 listening to requests. The oradb is the virtual path that maps to the OraDbUriServlet.

Note: In HTTP access special characters such as, ']', '[', '&', '|' have to be escaped using the %xxx format, where the xxx is the decimal number that points to the ASCII code for that character. Use the `getExternalUrl()` function in the `UriType` family to get the escaped URL version.

OraDBUriServlet Security

Consider these security issues when publishing `OraDbServlet`:

- Publishing `OraDBServlet` Under the `DBUser` realm
- Publishing `OraDBServlet` Under no realm

Publishing `OraDBServlet` Under the `DBUser` Realm

The `OraDBUri` servlet supplied, when published under the `DbUser` realm, automatically switches to the *authenticated* user and executes the query under that authenticated user.

For example:

- If the servlet is published as `SYS` under the `DBUser` realm, then the queries are always executed as the user who is logged in.
- If the servlet is published under the `Rdbms` realm, then the servlet is run as the publisher and not as the authenticated user.

Take care that the servlet is *not published under a realm other than the `DBUser` realm, particularly, if it is published in the `SYS` schema or if you want to enforce security to data access.*

Publishing `OraDBServlet` Under No Realm

If you do not publish the servlet under any realm, then the users can access the servlet directly without having to enter any username/password. The servlet executes under the privileges of the published user, and users of the servlet can access all data that the published user is privileged to see.

This can be useful in cases where you have a user that has restricted privileges and contains data (such as documents or demos or example schemas), which you would like any user to access.

For example, you can have a `HELP` user containing all helpful documentation related to a product or a company's operations and publish the servlet in that schema, giving access to everyone to access all the documents.

Special Note: If you publish the servlet under a realm other than the DbUserRealm, then queries are executed under the published user. So care **MUST** be taken to publish the servlet only in the DBUser realm, or if published under some other realm, the privileges granted to the published user must be limited. See the examples, starting with: "[DBUri Servlet Example 1: First Create a DBUriServer Web Service \[tkxmsrv.ssh\]](#)" on page 6-28.

Do not publish the servlet in any other realm other than the DBUser realm for the SYS user, otherwise users accessing the servlet will have privileges to access all your database data!

Installing OraDBUri Servlet

OraDbUriServlet is shipped in the jar file OraDbUri.jar under rdbms/jlib/ directory in your \$ORACLE_HOME. To install the servlet, perform the following tasks:

You can skip the first two steps, since the Java classes corresponding to the Java classes is already installed under the SYS user and execute privileges are granted to all the users. However, if you want the Java classes corresponding to the servlet to reside in your schema, then perform the following steps:

1. Run SQL*Plus and connect to the required schema that you want to install the servlet under. The preferred schema is the SYS schema.
2. Load the jar file in to the required schema by running the *inituris.sql* script found in the admin directory. You can also use loadjava to load the jar file into the schema. You need IO privileges to access files when running the dbms_java interface.
3. Now use the sess_sh to connect to your database at the admin port. See the *Oracle9i Oracle Servlet Engine User's Guide* for more details.
4. Set up the appropriate web services and realms. An example of setting up a DBUser realm is available under the servlet demos.
5. Once you have set up all the necessary realms and contexts, publish the servlet. For example in the sess_sh shell:

```
publishservlet -virtualpath /oradb/* -stateless  
/webdomains/default/contexts DBUriServlet SYS:oracle.xml.uri.OraDbUriServlet
```

- Note the use of `/oradb/*`. The `*` is necessary to indicate that any path following `oradb` is to be mapped to the same servlet. The `oradb` is published as the virtual path. Here, if you have installed the servlet in your own schema, then change the `SYS:` keyword to the schema name that the servlet is installed under.
- The `stateless` parameter indicates that the servlet itself is stateless and hence the same connection can be used to invoke the servlet.
- The `/webdomains/default/contexts` is the context under which the servlet is being published. Note that a context may have a virtual path already defined, in which case the virtual path that you publish the servlet under would be under that path. For example, if the context given here had a virtual path of `/servlets`, then the `DBUriServlet` can be accessed by the path `/servlets/oradb/*`.
- The `DBUriServlet` is the name of the servlet.
- The `oracle.xml.uri.OraDbUriServlet` is the name of the class to use under the `SYS` schema. Change the schema name to the schema under which the jar file was loaded.
- After this you can access the servlet directly through the Apache server using the `mod_ose` module or go directly to the OSE through the port on which the web service is listening.

DBUri Servlet Example 1: First Create a DBUriServer Web Service [tkxmsrv.ssh]

Script `tkxmsrv.ssh`, creates a `DBUriServer` Web service at port 8088 (hard coded in the script) and assigns ownership of the service to the user whose ID is passed as a parameter to the script. This script must be run before running any of the following scripts. In the environment in which the script was created, `AURORA_AWS_ADMIN_PORT` is set to 8080 and we have to connect to `AURORA_AWS_ADMIN_PORT` to do administrative activities. You can configure the Web service for any port by changing this part of the script.

```
# USAGE :
# sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh
<user-name>"
#
# This script does the following :
# 1. Creates dburiServer service with -root /dburidomain.
# 2. Assigns ownership to user passed as parameter

destroywebdomain dburidomain
```



```

destroyservice dburiServer

echo "Creating dburiService ..."
createwebservice -root /dburidomain dburiServer

#The following line requires this script to be run as SYS
rmendpoint -force dburiServer main
addendpoint -port 8088 -register dburiServer main

chown -R &l /dburidomain

echo "Service creation complete"

```

DBUri Servlet Example 2: Creating DBUridomain — Publishing OraDbUriServlet

This script creates webdomain under the service created by `tkxmsrv.ssh`. It publishes URI servlet in this domain using the default context. URI Servlet classes must be loaded under the SYS schema before running this script.

```

sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh SYS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsys.ssh"

# USAGE :
# sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsys.ssh"
# tkxmsrv.ssh must be run before running this script.
#
# This script does the following :
# 1. Creates webdomain /dburidomain.
# 2. Publishes the OraDbUriServlet servlet under SYS.

echo "Creating dburidomain ..."
createwebdomain /dburidomain

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>

echo "Domain creation complete"

echo "Publishing servlet under default context .."

publishservlet /dburidomain/contexts/default -virtualpath /norealm/*
DBUriServlet SYS:oracle.xml.dburi.OraDbUriServlet

```

```
echo "Servlet publishing complete .."
```

DBUri Servlet Example 3: Publishing OraDbUriServlet Under SYS [tkxmsysd.ssh]

This script creates a webdomain under the service created by `tkxmsrv.ssh`. It publishes URI servlet in this domain using the default context mapped to `DBUSER-realm`. URI Servlet classes must be loaded under `SYS` schema before running this script.

```
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh SYS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsysd.ssh"

#  USAGE :
#  sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsysd.ssh"
#  tkxmsrv.ssh must be run before running this script.
#
#  This script does the following :
#  1. Creates webdomain /dburidomain.
#  2. Creates and protects dburirealm
#  3. Publishes the OraDbUriServlet servlet under SYS.
#  4. Grant permission to execute the servlet to SCOTT and ADAMS.

echo "Creating dburidomain ..."
createwebdomain /dburidomain
#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>
echo "Domain creation complete"

echo "Creating and protecting dburirealm "
realm publish -w /dburidomain -r dburirealm -type DBUSER
realm publish -w /dburidomain -add dburirealm -type DBUSER
realm map -s /dburidomain/contexts/default -add /oradb/* -scheme
basic:dburirealm
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/default
-name SYS -path /oradb/* + get,post
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/default
-name SCOTT -path /oradb/* + get,post
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/default
-name ADAMS -path /oradb/* + get,post
echo "Realm creation complete"

echo "Publishing servlet under default context .."
```

```

publishservlet /dburidomain/contexts/default -virtualpath /oradb/* DBUriServlet
SYS:oracle.xml.dburi.OraDbUriServlet

chmod +x SCOTT /dburidomain/contexts/default/named_servlets/DBUriServlet

chmod +x ADAMS /dburidomain/contexts/default/named_servlets/DBUriServlet

echo "Servlet publishing complete .."

```

DBUri Servlet Example 4: Publishing OraDbUriServlet Under ADAMS

This script creates webdomain under the service created by `tkxmsrv.ssh`. It publishes URI servlet in this domain using the `uritests` context. URI Servlet classes must be loaded under `SYS` schema before running this script.

```

sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh
ADAMS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmadam.ssh"

#  USAGE :
#  sess_sh -s http://localhost:8080 -u adams/wood -c "@tkxmadam.ssh"
#  tkxmsrv.ssh must be run before running this script.
#
#  This script does the following :
#  1. Creates webdomain /dburidomain.
#  2. Creates uritests context.
#  3. Publishes the OraDbUriServlet servlet under ADAMS using the class
#     under SYS.

echo "Creating dburidomain and uritests context ..."
createwebdomain /dburidomain

echo "Creating uritests context ..."
createcontext -virtualpath /adamscon/ /dburidomain uritests

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/uritests -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/uritests -add /errors/internal -scheme <NONE>

echo "Domain creation complete"

```

```

echo "Publishing servlet under uritests context"

publishservlet /dburidomain/contexts/uritests -virtualpath /adamsdb/*
DBUriServlet SYS:oracle.xml.dburi.OraDbUriServlet

echo "Servlet publishing complete .."

```

DBUri Servlet Example 5: Publishing OraDbUriServlet Under SCOTT [tkxmsctd.ssh]

This script creates webdomain under the service created by tkxmsrv.ssh. It publishes URI servlet in this domain using the uritests context *mapped to DBUSER-realm*. URI Servlet classes must be loaded under SYS schema before running this script.

```

sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh
SCOTT"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsctd.ssh"

#  USAGE :
#  sess_sh -s http://localhost:8080 -u scott/tiger -c "@tkxmsctd.ssh"
#  tkxmsrv.ssh must be run before running this script.
#
#  This script does the following :
#  1. Creates webdomain /dburidomain.
#  2. Creates uritests context.
#  3. Creates and protects dburirealm
#  4. Publishes the OraDbUriServlet servlet under SCOTT.

echo "Creating dburidomain and uritests context ..."
createwebdomain /dburidomain

echo "Creating uritests context and granting ownership to SCOTT ..."
createcontext -virtualpath /scottcon/ /dburidomain uritests

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/uritests -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/uritests -add /errors/internal -scheme <NONE>

echo "Domain creation complete"

```

```

echo "Creating and protecting dburirealm "
realm publish -w /dburidomain -r dburirealm -type DBUSER
realm publish -w /dburidomain -add dburirealm -type DBUSER
realm map -s /dburidomain/contexts/uritests -add /scottdb/* -scheme
basic:dburirealm
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/uritests
-name PUBLIC -path /scottdb/* + get,post
echo "Realm creation complete"

echo "Publishing servlet under uritests context"

publishservlet /dburidomain/contexts/uritests -virtualpath /scottdb/*
DBUriServlet SYS:oracle.xml.dburi.OraDbUriServlet

echo "Servlet publishing complete .."

```

DBUri Servlet Example 6: Creating and Mapping dburirealm — OraDbUriServlet

This script creates webdomain under the service created by `tkxmsrv.ssh`. It publishes URI servlet in this domain using the default context mapped to RDBMS-realm. URI Servlet classes must be loaded under SYS schema before running this script.

```

sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh SYS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsysr.ssh"

#  USAGE :
#  sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsysr.ssh"
#  tkxmsrv.ssh must be run before running this script.
#
#  This script does the following :
#  1. Creates webdomain /dburidomain.
#  2. Creates and protects dburirealm using RDBMS realm mapping.
#  3. Publishes the OraDbUriServlet servlet under SYS.
#  4. Grant permission to execute the servlet to SCOTT and ADAMS.

echo "Creating dburidomain ..."
createwebdomain /dburidomain
#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>
echo "Domain creation complete"

```

```

echo "Creating and protecting dburirealm "

realm publish -w /dburidomain -r dburirealm -type rdbms
realm publish -w /dburidomain -add dburirealm -type rdbms
# create a user in the realm
realm user -w /dburidomain -realm dburirealm -add alex -p welcome
# create a group in the realm
realm group -w /dburidomain -realm dburirealm -add uriGroup -p welcome
# add 'alex' to the 'uriGroup'
realm parent -w /dburidomain -realm dburirealm -group uriGroup -add alex
# Allow 'uriGroup' to execute http requests with the GET,POST methods
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/default
-name uriGroup -path /rdbrealm/* + get,post
# protect the resource '/rdbrealm'
realm map -s /dburidomain/contexts/default -add /rdbrealm/* -scheme
Basic:dburirealm

echo "Realm creation complete"

echo "Publishing servlet under default context .."

publishservlet /dburidomain/contexts/default -virtualpath /rdbrealm/*
DBUriServlet SYS:oracle.xml.dburi.OraDbUriServlet

chmod +x SCOTT /dburidomain/contexts/default/named_servlets/DBUriServlet

echo "Servlet publishing complete .."

```

DBUri Servlet Example 7: Publishing OraDbUriServlet Under the ADAMS Schema

This script creates webdomain under the service created by `tkxmsrv.ssh`. It publishes URI servlet in this domain using `uritests` context. URI Servlet classes must be loaded under ADAMS schema before running this script.

```

sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh
ADAMS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmadmn.ssh"

#  USAGE :
#  sess_sh -s http://localhost:8080 -u adams/wood -c "@tkxmadmn.ssh"
#  tkxmsrv.ssh must be run before running this script.
#
#  This script does the following :
#  1. Creates webdomain /dburidomain.

```

```

# 2. Creates uritests context.
# 3. Publishes the OraDbUriServlet servlet under ADAMS using the class
#    under ADAMS.

echo "Creating dburidomain and uritests context ..."
createwebdomain /dburidomain

echo "Creating uritests context ..."
createcontext -virtualpath /adamscon/ /dburidomain uritests

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/uritests -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/uritests -add /errors/internal -scheme <NONE>

echo "Domain creation complete"

echo "Publishing servlet under uritests context"

publishservlet /dburidomain/contexts/uritests -virtualpath /adamsdb/*
DBUriServlet ADAMS:oracle.xml.dburi.OraDbUriServlet

echo "Servlet publishing complete .."

```

DBUri Servlet Example 8: Publishing OraDbUriServlet Under the ADAMS Schema

This script creates webdomain under the service created by `tkxmsrv.ssh`. It publishes URI servlet in this domain using uritests context mapped to DBUSER-realm. URI Servlet classes must be loaded under ADAMS schema before running this script.

```

sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmsrv.ssh
ADAMS"
sess_sh -s http://localhost:8080 -u sys/change_on_install -c "@tkxmadmd.ssh"

# USAGE :
# sess_sh -s http://localhost:8080 -u adams/wood -c "@tkxmadmd.ssh"
# tkxmsrv.ssh must be run before running this script.
#
# This script does the following :
# 1. Creates webdomain /dburidomain.

```

```
# 2. Creates uritests context.
# 3. Creates and protects dburirealm
# 4. Publishes the OraDbUriServlet servlet under ADAMS using the class
#    under ADAMS.

echo "Creating dburidomain and uritests context ..."
createwebdomain /dburidomain

echo "Creating uritests context and granting ownership to ADAMS ..."
createcontext -virtualpath /adamscon/ /dburidomain uritests

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/default -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/default -add /errors/internal -scheme <NONE>

#ensure that error pages are not protected
realm map -s /dburidomain/contexts/uritests -add /system/errors/* -scheme <NONE>
realm map -s /dburidomain/contexts/uritests -add /errors/internal -scheme <NONE>

echo "Domain creation complete"

echo "Creating and protecting dburirealm "
realm publish -w /dburidomain -r dburirealm -type DBUSER
realm publish -w /dburidomain -add dburirealm -type DBUSER
realm map -s /dburidomain/contexts/uritests -add /adamsdb/* -scheme
basic:dburirealm
realm perm -w /dburidomain -realm dburirealm -s /dburidomain/contexts/uritests
-name PUBLIC -path /adamsdb/* + get,post
echo "Realm creation complete"

echo "Publishing servlet under uritests context"

publishservlet /dburidomain/contexts/uritests -virtualpath /adamsdb/*
DBUriServlet ADAMS:oracle.xml.dburi.OraDbUriServlet

echo "Servlet publishing complete .."
```

Configuring the UriFactory Package to Handle DBUri-refs

The UriFactory, as explained earlier, "[UriFactory Package](#)" on page 6-17, if given an URL, would generate the appropriate subtypes of the UriType to handle the particular protocol. In the case of HTTP URLs, UriFactory would create instances of the HttpUriType. But, when you have a HTTP url which is really pointing into the

database using the DBUri-ref mechanism, then it would be more efficient to store and process it as a DBUriType instance in the database.

Inside the server, it is always more efficient to process the DBUri-ref directly using the DBUriType instances, instead of going through the HTTP URL mechanism. This is because the latter involves additional data transfer through the JavaVM, servlet, and web server layers, and could introduce additional character conversions.

If you have installed OraDBUriServlet to process the DBUri-refs, so that any URL such as `http://machine-name/servlets/oradb/` gets handled by that servlet, then you can configure the UriFactory to use that prefix and create instances of the DBUriType instead of the HttpUriType.

```
begin
  -- register a new handler for the dburi prefix..
  urifactory.registerHandler('http://machine-name/servlets/oradb'
    , 'SYS', 'DBURITYPE', true, true);
end;
/
```

Once you have executed this block in your session, then any `UriFactory.getUri()` call in that session, would automatically create an instance of the DBUriType for those HTTP URLs that have the prefix. In this way, you can convert the true DBUri URLs in to DBUriType instances for efficient processing.

XML SQL Utility (XSU)

This chapter contains the following sections:

- [What is XML SQL Utility \(XSU\)?](#)
- [XSU Dependencies and Installation](#)
- [XML SQL Utility and the Bigger Picture](#)
- [SQL-to-XML and XML-to-SQL Mapping Primer](#)
- [How XML SQL Utility Works](#)
- [Using the XSU Command Line Front End, OracleXML](#)
- [XSU Java API](#)
- [XSU PL/SQL API](#)
- [Advanced XSU Usage Techniques](#)
- [Frequently Asked Questions \(FAQs\): XML SQL Utility \(XSU\)](#)

What is XML SQL Utility (XSU)?

XML has become the format for data interchange. At the same time, a substantial amount of business data resides in object-relational databases. It is therefore necessary to have the ability to transform this relational data to XML.

XML SQL Utility (XSU) enables you to do this as follows:

- XSU can transform data retrieved from object-relational database tables or views into XML.
- XSU can extract data from an XML document, and using a canonical mapping, insert the data into appropriate columns or attributes of a table or a view.
- XSU can extract data from an XML document and apply this data to updating or deleting values of the appropriate columns or attributes.

Generating XML from the Database

For example, on the XML generation side, when given the query `SELECT * FROM emp`, XSU queries the database and returns the results as the following XML document:

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

Storing XML in the Database

Going the other way, given the XML document above, XSU can extract the data from it and insert it into the `scott.emp` table in the database.

Accessing XSU Functionality

XML SQL Utility functionality can be accessed in the following ways:

- Through a Java API
- Through a PL/SQL API
- Through a Java command line front end

See Also:

- [Appendix H, "XML SQL Utility \(XSU\) Specifications and Cheat Sheets"](#)
- *Oracle9i XML Reference*

XSU Features

XSU can perform the following tasks:

- Generate XML documents from any SQL query. XSU virtually supports all the datatypes supported in the Oracle database server.
- Dynamically generate DTDs (Document Type Definitions).
- During generation, perform simple transformations, such as modifying default tag names for the ROW element. You can also register an XSL transformation which is then applied to the generated XML documents as needed.
- Generate XML documents in their string or DOM representations.
- Insert XML into database tables or views. XSU can also update or delete records records from a database object, given an XML document.
- Easily generate complex nested XML documents. XSU can also store them in relational tables by creating object views over the flat tables and querying over these views. Object views can create structured data from existing relational data using Oracle8i and Oracle's object-relational infrastructure.

XSU Oracle Features

In Oracle, XSU can also perform the following tasks:

- Generates XML Schema given an SQL Query.
- Generates XML as a stream of SAX2 callbacks.
- Supports XML attributes during generation. This provides an easy way to specify that a particular column or group of columns should be mapped to an XML attribute instead of an XML element.

- SQL identifier to XML identifier escaping. Sometimes column names are not valid XML tag names. To avoid this you can either alias all the column names or turn on tag escaping.

Note: Oracle introduces the `DBMS_XMLGen` PL/SQL supplied package. This package provides the functionality previously available with `DBMS_XMLQuery`. `DBMS_XMLGen` is built into the database code, hence, it provides better performance.

XSU Dependencies and Installation

Dependencies

XML SQL Utility (XSU) needs the following components:

- **Database connectivity -- JDBC drivers.** XSU can work with any JDBC driver but is optimized for Oracle JDBC drivers. Oracle does not make any guarantee or provide support for the XSU running against non-Oracle databases.
- **XML Parser -- Oracle XML Parser, Version2.** Oracle XML Parser, version 2 is included in Oracle8i and Oracle, and is also available as part of the XSU install downloadable from the Oracle Technology Network (OTN) web site.

Installation

XML SQL Utility (XSU) is packaged with Oracle8i (8.1.7 and later) and Oracle. XSU is made up of three files:

- `$ORACLE_HOME/rdbms/jlib/xsu12.jar` -- Contains all the Java classes which make up XSU. `xsu12` requires `JDK1.2.x` and `JDBC2.x`. This is the XSU version loaded into Oracle.
- `$ORACLE_HOME/rdbms/jlib/xsu11.jar` -- Contains the same classes as `xsu12.jar`, except that `xsu11` requires `JDK1.1.x` and `JDBC1.x`.
- `$ORACLE_HOME/rdbms/admin/dbmsxsu.sql` -- This is the SQL script that builds the XSU PL/SQL API. `xsu12.jar` needs to be loaded into the database before `dbmsxsu.sql` is executed.

By default the Oracle installer installs XSU on the hard drive in the locations specified above. It also loads it into the database.

If during initial installation you choose to not install XSU, you can install it later, but the installation becomes less simple. To install XSU later, first install XSU and its dependent components on your system. You can accomplish this using Oracle Installer. Next perform the following steps:

1. If you have not yet loaded XML Parser for Java in the database, go to `$ORACLE_HOME/xdk/lib`. Here you will find `xmlparserv2.jar` that you need to load into the database. To do this, see “Loading JAVA Classes” in the *Oracle9i Java Stored Procedures Developer’s Guide*
2. Go to `$ORACLE_HOME/admin` and run `catxsu.sql`

Note: XML SQL Utility (XSU) is also available on OTN at: <http://otn.oracle.com/tech/xml> Check here for XSU updates.

XML SQL Utility and the Bigger Picture

XML SQL Utility (XSU) is written in Java, and can live in any tier that supports Java.

XML SQL Utility in the Database

The Java classes which make up XSU can be loaded into Java-enabled Oracle8i or later. Also, XSU contains a PL/SQL wrapper that publishes the XSU Java API to PL/SQL, creating a PL/SQL API. This way you can:

- Write new Java applications that run inside the database and that can directly access the XSU Java API
- Write PL/SQL applications that access XSU through its PL/SQL API
- Access XSU functionality directly through SQL

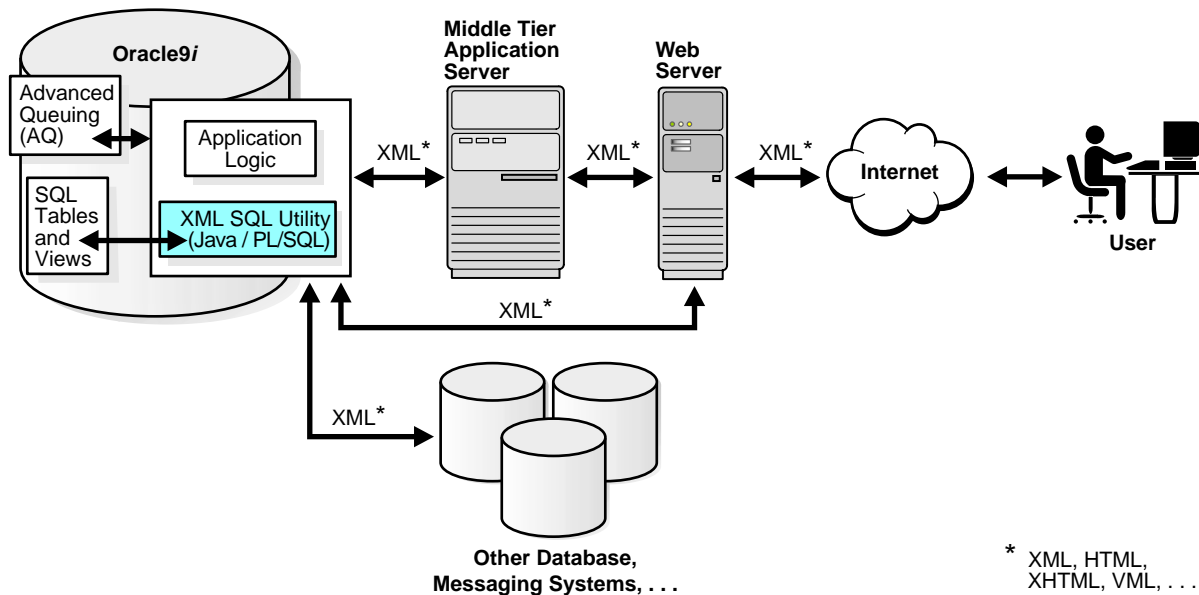
Note: To load and run Java code inside the database you need a Java-enabled Oracle8i or later server.

Figure 7-1 shows the typical architecture for such a system. XML generated from XSU running in the database, can be placed in advanced queues in the database to be queued to other systems or clients. The XML can be used from within stored

procedures in the database or shipped outside through web servers or application servers.

Note: In Figure 7–1, all lines are bi-directional. Since XSU can *generate* as well as *save* data, data can come from various sources to XSU running inside the database, and can be put back in the appropriate database tables.

Figure 7–1 Running XML SQL Utility in the Database



XML SQL Utility in the Middle Tier

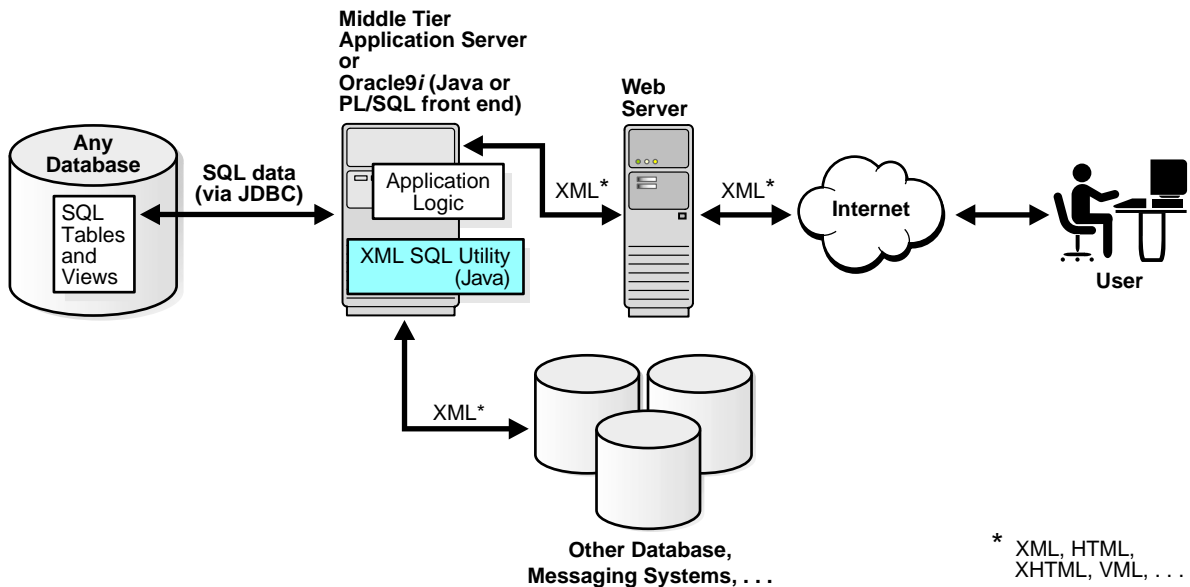
Your application architecture may need to use an application server in the middle tier, separate from the database. The application tier can be an Oracle database, Oracle Application Server, or a third party application server that supports Java programs.

You may want to generate XML in the middle tier, from SQL queries or ResultSets, for various reasons. For example, to integrate different JDBC data sources in the

middle tier. In this case you could install the XSU in your middle tier and your Java programs could make use of XSU through its Java API.

Figure 7-2, shows how a typical architecture for running XSU in a middle tier. In the middle tier, data from JDBC sources is converted by XSU into XML and then sent to Web servers or other systems. Again, the whole process is bi-directional and the data can be put back into the JDBC sources (database tables or views) using XSU. If an Oracle database itself is used as the application server, then you can also use the PL/SQL front-end instead of Java.

Figure 7-2 Running XML SQL Utility in the Middle Tier



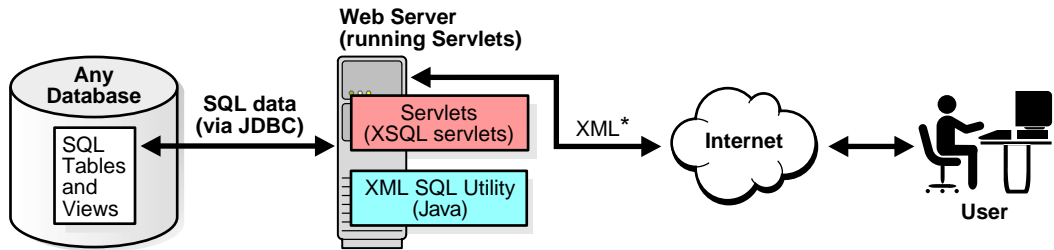
XML SQL Utility in a Web Server

XSU can live in the Web server, as long as the Web server supports Java servlets. This way you can write Java servlets that use XSU to accomplish their task.

XSQL servlet does just this. XSQL servlet is a standard servlet provided by Oracle. It is built on top of XSU and provides a template-like interface to XSU functionality. If XML processing in the Web server is your goal, you should probably use the XSQL servlet, as it will spare you from the intricate servlet programming.

See: [Chapter 10, "XSQL Pages Publishing Framework"](#) for information about using XSQL Servlet.

Figure 7-3 Running XML SQL Utility in a Web Server



* XML, HTML, XHTML, VML, . . .

XML SQL Utility In The Client Tier

XML SQL Utility can be also installed on a client system, where you can write Java programs that use XSU. You can also use XSU directly through its command line front end.

SQL-to-XML and XML-to-SQL Mapping Primer

As described earlier, XML SQL Utility transforms data retrieved from object-relational database tables or views into XML. XSU can also extract data from an XML document, and using a specified mapping, insert the data into appropriate columns or attributes of a table or a view in the database. This section describes the canonical mapping or transformation used to go from SQL to XML or vice versa.

Default SQL-to-XML Mapping

Consider table emp:

```
CREATE TABLE emp
(
  EMPNO NUMBER,
  ENAME VARCHAR2(20),
  JOB VARCHAR2(20),
  MGR NUMBER,
```

```

        HIREDATE DATE,
        SAL NUMBER,
        DEPTNO NUMBER
    );

```

XSU can generate the following XML document by specifying the query, `select * from emp;`

```

<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>

```

In the generated XML, the rows returned by the SQL query are enclosed in a `ROWSET` tag to constitute the `<ROWSET>` element. This element is also the root element of the generated XML document.

- The `<ROWSET>` element contains one or more `<ROW>` elements.
- Each of the `<ROW>` elements contain the data from one of the returned database table rows. Specifically, each `<ROW>` element contains one or more elements whose names and content are those of the database columns specified in the `SELECT` list of the SQL query.
- These elements, corresponding to database columns, contain the data from the columns.

SQL-to-XML Mapping Against Object-Relational Schema

Next we describe this mapping but against an object-relational schema. Consider the following type, `AddressType`. Its an object type whose attributes are all scalar types and is created as follows:

```

CREATE TYPE AddressType AS OBJECT (
    STREET VARCHAR2(20),
    CITY   VARCHAR2(20),
    STATE  CHAR(2),

```

```
        ZIP    VARCHAR2(10)
    );
/
```

The following type, `EmployeeType`, is also an object type but it has an `EMPADDR` attribute that is of an object type itself, specifically, `AddressType`. `EmployeeType` is created as follows:

```
CREATE TYPE EmployeeType AS OBJECT
(
    EMPNO NUMBER,
    ENAME VARCHAR2(20),
    SALARY NUMBER,
    EMPADDR AddressType
);
/
```

The following type, `EmployeeListType`, is a collection type whose elements are of the object type, `EmployeeType`. `EmployeeListType` is created as follows:

```
CREATE TYPE EmployeeListType AS TABLE OF EmployeeType;
/
```

Finally, `dept` is a table with, among other things, an object type column and a collection type column -- `AddressType` and `EmployeeListType` respectively.

```
CREATE TABLE dept
(
    DEPTNO NUMBER,
    DEPTNAME VARCHAR2(20),
    DEPTADDR AddressType,
    EMPLIST EmployeeListType
)
NESTED TABLE EMPLIST STORE AS EMPLIST_TABLE;
```

Assume that valid values are stored in table, `dept`. For the query `select * from dept`, XSU generates the following XML document:

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <DEPTNO>100</DEPTNO>
    <DEPTNAME>Sports</DEPTNAME>
    <DEPTADDR>
      <STREET>100 Redwood Shores Pkwy</STREET>
      <CITY>Redwood Shores</CITY>
```

```

    <STATE>CA</STATE>
    <ZIP>94065</ZIP>
  </DEPTADDR>
  <EMPLIST>
    <EMPLIST_ITEM num="1">
      <EMPNO>7369</EMPNO>
      <ENAME>John</ENAME>
      <SALARY>10000</SALARY>
      <EMPADDR>
        <STREET>300 Embarcadero</STREET>
        <CITY>Palo Alto</CITY>
        <STATE>CA</STATE>
        <ZIP>94056</ZIP>
      </EMPADDR>
    </EMPLIST_ITEM>
    <!-- additional employee types within the employee list -->
  </EMPLIST>
</ROW>
<!-- additional rows ... -->
</ROWSET>

```

As in the last example, the mapping is canonical, that is, <ROWSET> contains <ROW>s that contain elements corresponding to the columns. As before, the elements corresponding to scalar type columns simply contain the data from the column.

Mapping Complex Type Columns to XML

Things get more complex with elements corresponding to a complex type column. For example, <DEPTADDR> corresponds to the DEPTADDR column which is of object type ADDRESS. Consequently, <DEPTADDR> contains subelements corresponding to the attributes specified in the type ADDRESS. These subelements can contain data or sub-elements of their own, again depending if the attribute they correspond to is of a simple or complex type.

Mapping Collections to XML

When dealing with elements corresponding to database collections, things are yet different. Specifically, the <EMPLIST> element corresponds to the EMPLIST column which is of a EmployeeListType collection type. Consequently, the <EMPLIST> element contains a list of <EMPLIST_ITEM> elements each corresponding to one of the elements of the collection.

Other observations to make about the above mapping are:

- The <ROW> elements contains a cardinality attribute num.
- If a particular column or attribute value is null, then for that row, the corresponding XML element is left out altogether.
- If a top level scalar column name starts with the at sign (@) character, then the particular column is mapped to an XML attribute instead of an XML element.

Customizing the Generated XML: Mapping SQL to XML

Often, one needs to generate XML with a specific structure. Since the desired structure may differ from the default structure of the generated XML document, it is desirable to have some flexibility in this process. You can customize the structure of a generated XML document using one of the following methods:

- "Source Customization"
- "Mapping Customization"
- "Post-Generation Customization"

Source Customization

Source customizations are done by altering the query or database schema. The simplest and most powerful source customizations include the following:

- ***In the database schema***, create an object-relational view that maps to the desired XML document structure.
- ***In your query***:
 - Use cursor subqueries, or cast-multiset constructs to get nesting in the XML document which comes from a flat schema.
 - Alias column/attribute names to get the desired XML element names.
 - Alias top level scalar type columns with identifiers which begin with the at sign (@) to have them map to an XML attribute instead of an XML element . For example, `select empno as "@empno", ... from emp`, results in an XML document where the <ROW> element has an attribute EMPNO.

Mapping Customization

XML SQL Utility allows you to modify the mapping it uses to transform SQL data into XML. You can make any of the following SQL to XML mapping changes:

- Change or omit the <ROWSET> tag.

- Change or omit the `<ROW>` tag.
- Change or omit the attribute `num`. This is the cardinality attribute of the `<ROW>` element.
- Specify the case for the generated XML element names.
- Specify that XML elements corresponding to elements of a collection, should have a cardinality attribute.
- Specify the format for dates in the XML document.
- Specify that null values in the XML document should be indicated using a nullness attribute, rather than by omission of the element.

Post-Generation Customization

Finally, if the desired customization cannot be achieved with the foregoing methods, you can write an XSL transformation and register it with XSU. While there is an XSLT registered with the XSU, XSU can apply XSLT to any XML it generates.

Default XML-to-SQL Mapping

XML to SQL mapping is just the reverse of the SQL to XML mapping.

See Also: ["Default SQL-to-XML Mapping"](#) on page 7-8.

Consider the following differences when mapping from XML to SQL, compared to mapping from SQL to XML:

- When going from XML to SQL, the XML attributes are ignored. Thus, there is really no mapping of XML attributes to SQL.
- When going from SQL to XML, mapping is performed from the resultset created by the SQL query to XML. This way the query can span multiple database tables or views. What gets formed is a single resultset which is then converted into XML. This is not the case when going from XML to SQL, where:
 - To insert one XML document into multiple tables or views, you must create an object-relational view over the target schema.
 - If the view is not updatable, one work around is to use `INSTEAD-OF-INSERT` triggers.

If the XML document does not perfectly map into the target database schema, there are three things you can do:

- **Modify the Target.** Create an object-relational view over the target schema, and make the view the new target.
- **Modify the XML Document.** Use XSLT to transform the XML document. The XSLT can be registered with XSU so that the incoming XML is automatically transformed, before any mapping attempts are made. This is the least performant solution.
- **Modify XSU's XML-to-SQL Mapping.** You can instruct XSU to perform case insensitive matching of the XML elements to database columns or attributes.
 - If not the default (ROW), you can tell XSU to use the name of the element corresponding to a database row.
 - You can instruct XSU on which date format to use when parsing dates in the XML document.

How XML SQL Utility Works

This section describes how XSU works when performing the following tasks:

- [Selecting with XSU](#) on page 7-14
- [Inserting with XSU](#) on page 7-14
- [Updating with XSU](#) on page 7-15
- [Deleting with XSU](#) on page 7-16

Selecting with XSU

XSU generation is simple. SQL queries are executed and the resultset is retrieved from the database. Metadata about the resultset is aquired and analyzed. Using the mapping described in "[Default SQL-to-XML Mapping](#)" on page 7-8, the SQL result set is processed and converted into an XML document.

Inserting with XSU

To insert the contents of an XML document into a particular table or view, XSU first retrieves the metadata about the target table or view. Based on the metadata, XSU generates an SQL INSERT statement. XSU extracts the data out of the XML document and binds it to the appropriate columns or attributes. Finally the statement is executed.

For example, assume that the target table is `dept` and the XML document is the one generated from `dept`.

See Also: ["Default SQL-to-XML Mapping"](#) on page 7-8.

XSU generates the following `INSERT` statement.

```
INSERT INTO Dept (DEPTNO, DEPTNAME, DEPTADDR, EMPLIST) VALUES (?, ?, ?, ?)
```

Next, the XSU parses the XML document, and for each record, it binds the appropriate values to the appropriate columns or attributes, and executes the statement:

```
DEPTNO <- 100
DEPTNAME <- SPORTS
DEPTADDR <- AddressType('100 Redwood Shores Pkwy', 'Redwood Shores',
                        'CA', '94065')

EMPLIST <- EmployeeListType(EmployeeType(7369, 'John', 100000,
                                         AddressType('300 Embarcadero', 'Palo Alto', 'CA', '94056')), ...)
```

Insert processing can be optimized to insert in batches, and commit in batches. More detail on batching can be found in the section on ["Insert Processing Using XSU \(Java API\)"](#) on page 7-36.

Updating with XSU

Updates and deletes differ from inserts in that they can affect more than one row in the database table. For inserts, each `ROW` element of the XML document can affect at most, one row in the table, provided that there are no triggers or constraints on the table.

However, with both updates and deletes, the XML element could match more than one row if the matching columns are not key columns in the table. For updates, you must provide a list of key columns which XSU needs to identify the row to update. For example, to update the `DEPTNAME` to `SportsDept` instead of `Sports`, you can have an XML document such as:

```
<ROWSET>
  <ROW num="1">
    <DEPTNO>100</DEPTNO>
    <DEPTNAME>SportsDept</DEPTNAME>
  </ROW>
</ROWSET>
```

and supply the DEPTNO as the key column. This would result in the following UPDATE statement:

```
UPDATE DEPT SET DEPTNAME = ? WHERE DEPTNO = ?
```

and bind the values,

```
DEPTNO <- 100
DEPTNAME <- SportsDept
```

For updates, you can also choose to update only a set of columns and not all the elements present in the XML document. See also, "[Update Processing Using XSU \(Java API\)](#)" on page 7-38.

Deleting with XSU

For deletes, you can choose to give a set of key columns for the delete to identify the rows. If the set of key columns are not given, then the DELETE statement tries to match all the columns given in the document. For an XML document:

```
<ROWSET>
  <ROW num="1">
    <DEPTNO>100</DEPTNO>
    <DEPTNAME>Sports</DEPTNAME>
    <DEPTADDR>
      <STREET>100 Redwood Shores Pkwy</STREET>
      <CITY>Redwood Shores</CITY>
      <STATE>CA</STATE>
      <ZIP>94065</ZIP>
    </DEPTADDR>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

To delete, XSU fires off a DELETE statement (one per ROW element) which looks like the following:

```
DELETE FROM Dept WHERE DEPTNO = ? AND DEPTNAME = ? AND DEPTADDR = ?
binding,
DEPTNO <- 100
DEPTNAME <- Sports
DEPTADDR <- AddressType('100 Redwood Shores Pkwy', 'Redwood
City', 'CA', '94065')
```

See also, "[Delete Processing Using XSU \(Java API\)](#)" on page 7-41.

Using the XSU Command Line Front End, OracleXML

XSU comes with a simple command line front end which gives you quick access to XML generation and insertion. In Oracle, the XSU front end does not publish the update and delete functionalities. The XSU command line options are provided through the Java class, `OracleXML`. Invoke it by calling:

```
java OracleXML
```

This prints the front end usage information. To run the XSU command line front end, first specify where the executable is located. Add the following to your `CLASSPATH`:

- XSU Java library (`xsu12.jar` or `xsu11.jar`)

Also, since XSU depends on Oracle XML Parser and JDBC drivers, make the location of these components known. To do this, the `CLASSPATH` must include the locations of:

- Oracle XML Parser Java library (`xmlparserv2.jar`)
- JDBC library (`classes12.jar` if using `xsu12.jar` or `classes11.jar` if using `xsu11.jar`)

Generating XML Using the XSU Command Line

For XSU generation capabilities, use the XSU `getXML` parameter. For example, to generate an XML document by querying the `emp` table in the `scott` schema, use:

```
java OracleXML getXML -user "scott/tiger" "select * from emp"
```

This performs the following tasks:

- Connects to the current default database
- Executes the query `select * from emp`
- Converts the result to XML
- Displays the result

The `getXML` parameter supports a wide range of options which are explained in the following section.

XSU's OracleXML getXML Options

Table 7-1 lists the OracleXML getXML options:

Table 7-1 XSU's OracleXML getXML Options

getXML Option	Description
-user "<username>/<password>"	Specifies the user name and password to connect to the database. If this is not specified, the user defaults to <code>scott/tiger</code> . Note that the connect string is also being specified, the user name and password can be specified as part of the connect string.
-conn "<JDBC_connect_string>"	Specifies the JDBC database connect string. By default the connect string is: <code>"jdbc:oracle:oci8:@"</code> .
-withDTD	Instructs the XSU to generate the DTD along with the XML document.
-rowsetTag "<tag_name>"	Specifies <code>rowset</code> tag (the tag that encloses all the XML elements corresponding to the records returned by the query). The default <code>rowset</code> tag is <code>ROWSET</code> . Specifying an empty string for the <code>rowset</code> tells the XSU to completely omit the <code>rowset</code> element.
-rowTag "<tag_name>"	Specifies the <code>row</code> tag (the tag used to enclose the data corresponding to a database row). The default <code>row</code> tag is <code>ROW</code> . Specifying an empty string for the <code>row</code> tag tells the XSU to completely omit the <code>row</code> tag.
-rowIdAttr "<row_id-attribute-name>"	Names the attribute of the <code>ROW</code> element keeping track of the cardinality of the <code>rows</code> . By default this attribute is called <code>num</code> . Specifying an empty string (i.e. <code>""</code>) as the <code>rowID</code> attribute will tell the XSU to omit the attribute.
-rowIdColumn "<row Id column name>"	Specifies that the value of one of the scalar columns from the query should be used as the value of the <code>rowID</code> attribute.
-collectionIdAttr "<collection id attribute name>"	Names the attribute of an XML list element keeping track of the cardinality of the elements of the list (Note: the generated XML lists correspond to either a cursor query, or collection). Specifying an empty string (i.e. <code>""</code>) as the <code>rowID</code> attribute will tell the XSU to omit the attribute.
-useNullAttrId	Tells the XSU to use the attribute <code>NULL (TRUE/FALSE)</code> to indicate the nullness of an element.
-styleSheet "<stylesheet URI>"	Specifies the stylesheet in the XML PI (Processing Instruction).
-stylesheetType "<stylesheet type>"	Specifies the stylesheet type in the XML PI (Processing Instruction).
-errorTag "<error tag name>"	Specifies the error tag -- the tag to enclose error messages which are formatted into XML.

Table 7-1 XSU's OracleXML getXML Options (Cont.)

getXML Option	Description
-raiseNoRowsException	Tells the XSU to raise an exception if no rows are returned.
-maxRows "<maximum number of rows>"	Specifies the maximum number of rows to be retrieved and converted to XML.
-skipRows "<number of rows to skip>"	Specifies the number of rows to be skipped.
-encoding "<encoding name>"	Specifies the character set encoding of the generated XML.
-dateFormat "<date format>"	Specifies the date format for the date values in the XML document.
-fileName "<SQL query fileName>" <sql query>	Specifies the file name which contains the query or specify the query itself.

Inserting XML Using XSU's Command Line (putXML)

To insert an XML document into the `emp` table in the `scott` schema, use the following syntax:

```
java OracleXML putXML -user "scott/tiger" -fileName "/tmp/temp.xml" "emp"
```

This performs the following tasks:

- Connects to the current database
- Reads the XML document from the given file
- Parses it, matches the tags with column names
- Inserts the values appropriately in to the `emp` table

Note: The XSU command line front end, `putXML`, currently only publishes XSU `insert` functionality. It may be expanded in future to also publish XSU `update` and `delete` functionality.

XSU OracleXML putXML Options

Table 7-2 lists the putXML options:

Table 7-2 XSU's OracleXML putXML Options

putXML Options	Description
-user "<username>/<password>"	Specifies the user name and password to connect to the database. If this is not specified, the user defaults to <code>scott/tiger</code> . Note that the connect string is also being specified, the user name and password can be specified as part of the connect string.
-conn "<JDBC_connect_string>"	Specifies the JDBC database connect string. By default the connect string is: <code>"jdbc:oracle:oci8:@"</code> .
-batchSize "<batching size>"	Specifies the batch size, which control the number of rows which are batched together and inserted in a single trip to the database. Batching improves performance.
-commitBatch "<commit size>"	Specifies the number of inserted records after which a commit is to be executed. Note that if the <code>autocommit</code> is <code>true</code> (default), then setting the <code>commitBatch</code> has no consequence.
-rowTag "<tag_name>"	Specifies the <code>row</code> tag (the tag used to enclose the data corresponding to a database row). The default row tag is <code>ROW</code> . Specifying an empty string for the <code>row</code> tag tells the XSU that no row enclosing tag is used in the XML document.
-dateFormat "<date format>"	Specifies the date format for the date values in the XML document.
-ignoreCase	Makes the matching of the column names with tag names case insensitive (e.g. "EmpNo" will match with "EMPNO" if <code>ignoreCase</code> is on).
-fileName "<file name>" -URL "<url>" -xmlDoc "<xml document>"	Specifies the XML document to insert. The <code>fileName</code> option specifies a local file, the <code>URL</code> specifies a URL to fetch the document from and the <code>xmlDoc</code> option inlines the XML document as a string on the command line.
<tableName>	The name of the table to put the values into.

XSU Java API

The following two classes make up the XML SQL Utility Java API:

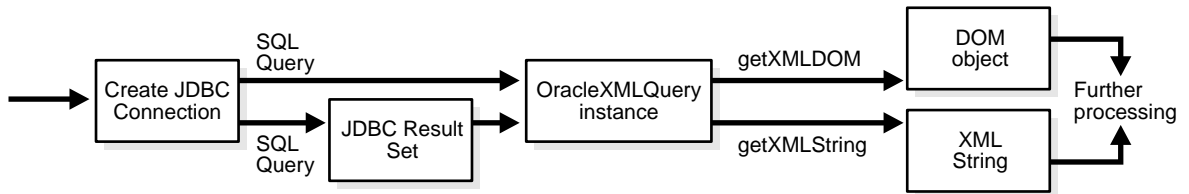
- XSU API for XML generation: `oracle.xml.sql.query.OracleXMLQuery`
- XSU API for XML save, insert, update, and delete:
`oracle.xml.sql.dml.OracleXMLSave`

Generating XML with XSU's OracleXMLQuery

The `OracleXMLQuery` class makes up the XML generation part of the XSU Java API. [Figure 7-4](#) illustrates the basic steps you need to take when using `OracleXMLQuery` to generate XML:

1. Create a connection.
2. Create an `OracleXMLQuery` instance by supplying an SQL string or a `ResultSet` object.
3. Obtain the result as a DOM tree or XML string.

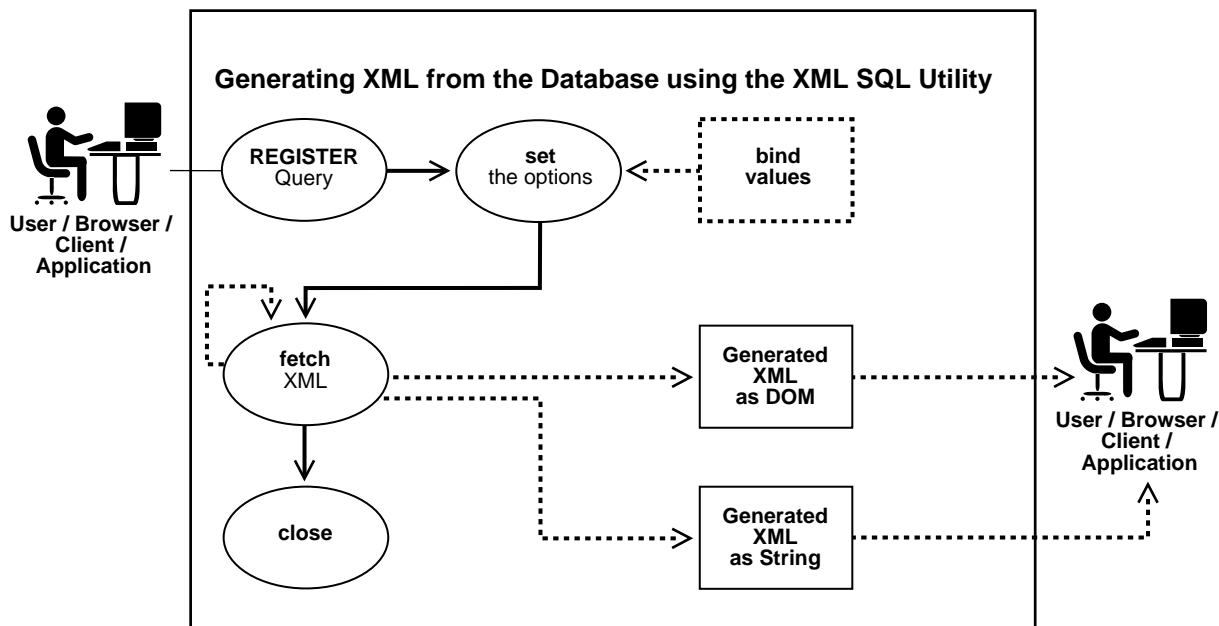
Figure 7-4 *Generating XML With XML SQL Utility for Java: Basic Steps*



Generating XML From SQL Queries Using XSU

The following examples illustrate how XSU can generate an XML document in its DOM or string representation given a SQL query. See [Figure 7-5](#).

Figure 7-5 Generating XML With XML SQL Utility



XSU Generating XML Example 1: Generating a String From Table emp (Java)

1. Create a connection

Before generating the XML you must create a connection to the database. The connection can be obtained by supplying the JDBC connect string. First register the Oracle JDBC class and then create the connection, as follows

```
// import the Oracle driver..
import oracle.jdbc.driver.*;

// Load the Oracle JDBC driver
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

// Create the connection.
Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci8:@", "scott", "tiger");
```

Here, the connection is done using OCI8's JDBC driver. You can connect to the scott schema supplying the password tiger. It connects to the current

database (identified by the `ORA_SID` environment variable). You can also use the JDBC thin driver to connect to the database. The thin driver is written in pure Java and can be called from within applets or any other Java program.

See Also: *Oracle9i Java Developer's Guide* for more details.

- **Connecting With the Thin Driver.** Here is an example of connecting using the JDBC thin driver:

```
// Create the connection.
Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@dlsun489:1521:ORCL",
                             "scott","tiger");
```

The thin driver requires you to specify the host name (`dlsun489`), port number (`1521`), and the Oracle SID (`ORCL`), which identifies a specific Oracle instance on the machine.

- **No Connection Needed When Run In the Server.** When writing server side Java code, that is, when writing code that will run on the server, you need not establish a connection using a username and password, since the server-side internal driver runs within a default session. You are already connected. In this case call the `defaultConnection()` on the `oracle.jdbc.driver.OracleDriver()` class to get the current connection, as follows:

```
import oracle.jdbc.driver.*;

// Load the Oracle JDBC driver
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
Connection conn = new oracle.jdbc.driver.OracleDriver
().defaultConnection ();
```

The remaining discussion either assumes you are using an OCI8 connection from the client or that you already have a connection object created. Use the appropriate connection creation based on your needs.

2. Creating an OracleXMLQuery Class Instance

Once you have registered your connection, create an `OracleXMLQuery` class instance by supplying a SQL query to execute as follows:

```
// import the query class in to your class
import oracle.xml.sql.query.OracleXMLQuery;
```

```
OracleXMLQuery qry = new OracleXMLQuery (conn, "select * from emp");
```

You are now ready to use the query class.

3. Obtain the result as a DOM tree or XML string

- **DOM Object Output.** If, instead of a string, you wanted a DOM object, you can simply request a DOM output as follows:

```
org.w3c.DOM.Document domDoc = qry.getXMLDOM();
```

and use the DOM traversals.

- **XML String Output.** You can get an XML string for the result by:

```
String xmlString = qry.getXMLString();
```

Here is a complete listing of the program to extract (generate) the XML string. This program gets the string and prints it out to standard output:

```
import oracle.jdbc.driver.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.lang.*;
import java.sql.*;

// class to test the String generation!
class testXMLSQL {

    public static void main(String[] argv)
    {

        try{
            // create the connection
            Connection conn = getConnection("scott","tiger");

            // Create the query class.
            OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp");

            // Get the XML string
            String str = qry.getXMLString();

            // Print the XML output
            System.out.println(" The XML output is:\n"+str);
            // Always close the query to get rid of any resources..
            qry.close();
        }catch(SQLException e){
            System.out.println(e.toString());
        }
    }
}
```

```
    }  
  }  
  
  // Get the connection given the user name and password..!  
  private static Connection getConnection(String username, String password)  
    throws SQLException  
  {  
    // register the JDBC driver..  
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  
  
    // Create the connection using the OCI8 driver  
    Connection conn =  
      DriverManager.getConnection("jdbc:oracle:oci8:@",username,password);  
  
    return conn;  
  }  
}
```

How to Run This Program

To run this program, carry out the following:

1. Store this in a file called `testXMLSQL.java`
2. Compile it using `javac`, the Java compiler
3. Execute it by specifying: `java testXMLSQL`

You must have the `CLASSPATH` pointing to this directory for the Java executable to find the class. Alternatively use various visual Java tools including Oracle JDeveloper to compile and run this program. When run, this program prints out the XML file to the screen.

XSU Generating XML Example 2: Generating DOM From emp table (Java)

DOM (Document Object Model) is a standard defined by the W3C committee. DOM represents an XML document in a parsed tree-like form. Each XML entity becomes a DOM node. Thus XML elements and attributes become DOM nodes while their children become child nodes. To generate a DOM tree from the XML generated by XSU, you can directly request a DOM document from XSU, as it saves the overhead of having to create a string representation of the XML document and then parse it to generate the DOM tree.

XSU calls the parser to directly construct the DOM tree from the data values. The following example illustrates how to generate a DOM tree. The example steps through the DOM tree and prints all the nodes one by one.

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;

class domTest{

    public static void main(String[] argv)
    {
        try{
            // create the connection
            Connection conn = getConnection("scott","tiger");

            // Create the query class.
            OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp");

            // Get the XML DOM object. The actual type is the Oracle Parser's DOM
            // representation. (XMLDocument)
            XMLDocument domDoc = (XMLDocument)qry.getXMLDOM();

            // Print the XML output directly from the DOM
            domDoc.print(System.out);

            // If you instead want to print it to a string buffer you can do
            this..!
            StringWriter s = new StringWriter(10000);
            domDoc.print(new PrintWriter(s));
            System.out.println(" The string version ----> "+s.toString());

            qry.close(); // You should always close the query!!
        }catch(Exception e){
            System.out.println(e.toString());
        }
    }

    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
        throws SQLException
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

```
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
    return conn;
    }
}
```

Paginating Results: skipRows and maxRows

In the examples shown so far, XML SQL Utility (XSU) takes the `ResultSet` or the query and generates the whole document from all the rows of the query. To obtain 100 rows at a time, you would then have to fire off different queries to get the first 100 rows, the next 100, and so on. Also it is not possible to skip the first five rows of the query and then generate the result.

To obtain the desired results, use the XSU `skipRows` and `maxRows` parameter settings:

- `skipRows` parameter, when set, forces the generation to skip the desired number of rows before starting to generate the result.
- `maxRows` limits the number of rows converted to XML.

For example, if you set `skipRows` to a value of 5 and `maxRows` to a value of 10, then XSU skips the first 5 rows, then generates XML for the next 10 rows.

Keeping the Object Open For the Duration of the User's Session

In Web scenarios, you may want to keep the query object open for the duration of the user's session. For example, consider the case of a Web search engine which gives the results of a user's search in a paginated fashion. The first page lists 10 results, the next page lists 10 more results, and so on.

To achieve this, request XSU to convert 10 rows at a time and keep the `ResultSet` state alive, so that the next time you ask XSU for more results, it starts generating from the place the last generation finished. See ["XSU Generating XML Example 3. Paginating Results: Generating an XML Page \(Java\)"](#) on page 7-28.

When the Number of Rows or Columns in a Row Are Too Large

There is also the case when the number of rows, or number of columns in a row are very large. In this case, you can generate multiple documents each of a smaller size.

These cases can be handled by using the `maxRows` parameter and the `keepObjectOpen` function.

keepObjectOpen Function

Typically, as soon as all results are generated, `OracleXMLQuery` internally closes the `ResultSet`, if it created one using the SQL query string given, since it assumes you no longer want any more results. However, in the case described above, to maintain that state, you need to call the `keepObjectOpen` function to keep the cursor alive. See the following example.

XSU Generating XML Example 3. Paginating Results: Generating an XML Page (Java)

This example, writes a simple class that maintains the state and generates the next page each time it is called.

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;
public class pageTest
{
    Connection conn;
    OracleXMLQuery qry;
    ResultSet rset;
    Statement stmt;
    int lastRow = 0;

    public pageTest(String sqlQuery)
    {
        try{
            conn = getConnection("scott","tiger");
            //stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            //                          ResultSet.CONCUR_READ_ONLY); // create a scrollable Rset
            //stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
            //                          ResultSet.CONCUR_READ_ONLY); // create a scrollable Rset
            stmt = conn.createStatement();
            ResultSet rset = stmt.executeQuery(sqlQuery); // get the result set..
            rset.first();
            qry = new OracleXMLQuery(conn,rset); // create a OracleXMLQuery instance
            qry.keepCursorState(true); // Don't lose state after the first fetch
            qry.setRaiseNoRowsException(true);
            qry.setRaiseException(true);
        }
    }
}
```

```
    }catch(SQLException e){
        System.out.println(e.toString());
    }
}

// Returns the next XML page..!
public String getResult(int startRow, int endRow) throws SQLException
{
    //rset.relative(lastRow-startRow); // scroll inside the result set
    //rset.absolute(startRow); // scroll inside the result set
    qry.setMaxRows(endRow-startRow); // set the max # of rows to retrieve..!
    //System.out.println("before getxml");
    return qry.getXMLString();
}

// Function to still perform the next page.
public String nextPage() throws SQLException
{
    String result = getResult(lastRow,lastRow+10);
    lastRow+= 10;
    return result;
}

public void close() throws SQLException
{
    stmt.close(); // close the statement..
    conn.close(); // close the connection
    qry.close(); // close the query..
}

public static void main(String[] argv)
{
    String str;

    try{
        pageTest test = new pageTest("select e.* from emp e");

        int i = 0;
        // Get the data one page at a time..!!!!
        while ((str = test.getResult(i,i+10))!= null)
        {
            System.out.println(str);
            i+= 10;
        }
        test.close();
    }
}
```

```
        }catch(Exception e){
            e.printStackTrace(System.out);
        }
    }
}
// Get the connection given the user name and password..!
private static Connection getConnection(String user, String passwd)
    throws SQLException
{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
    return conn;
}
}
```

Generating XML from ResultSet Objects

You saw how you can supply a SQL query and get the results as XML. In the last example, you retrieved paginated results. However in Web cases, you may want to retrieve the previous page and not just the next page of results. To provide this scrollable functionality, you can use the `Scrollable ResultSet`. Use the `ResultSet` object to move back and forth within the result set and use XSU to generate the XML each time. The following example illustrates how to do this.

XSU Generating XML Example 4: Generating XML from JDBC ResultSets (Java)

This example shows you how to use the JDBC `ResultSet` to generate XML. Note that using the `ResultSet` might be necessary in cases that are not handled directly by XSU, for example, when setting the batch size, binding values, and so on. This example extends the previously defined `pageTest` class to handle any page.

```
public class pageTest()
{
    Connection conn;
    OracleXMLQuery qry;
    ResultSet rset;
    int lastRow = 0;

    public pageTest(String sqlQuery)
    {
        conn = getConnection("scott","tiger");
        Statement stmt = conn.createStatement(sqlQuery); // create a scrollable Rset
        ResultSet rset = stmt.executeQuery(); // get the result set..
    }
}
```



```
        qry = new OracleXMLQuery(conn,rset);    // create a OracleXMLQuery instance
        qry.keepObjectOpen(true); // Don't lose state after the first fetch
    }

    // Returns the next XML page..!
    public String getResult(int startRow, int endRow)
    {
        rset.scroll(lastRow-startRow); // scroll inside the result set
        qry.setMaxRows(endRow-startRow); // set the max # of rows to retrieve..!
        return qry.getXMLString();
    }

    // Function to still perform the next page.
    public String nextPage()
    {
        String result = getResult(lastRow,lastRow+10);
        lastRow+= 10;
        return result;
    }

    public void close()
    {
        stmt.close();    // close the statement..
        conn.close();    // close the connection
        qry.close();    // close the query..
    }

    public void main(String[] argv)
    {
        pageTest test = new pageTest("select * from emp");

        int i = 0;
        // Get the data one page at a time..!!!!
        while ((str = test.getResult(i,i+10))!= null)
        {
            System.out.println(str);
            i+= 10;
        }
        test.close();
    }
}
```

XSU Generating XML Example 5: Generating XML from Procedure Return Values

The `OracleXMLQuery` class provides XML conversion only for query strings or `ResultSets`. But in your application if you have PL/SQL procedures that return REF cursors, how would you do the conversion?

In this case, you can use the abovementioned `ResultSet` conversion mechanism to perform the task. REF cursors are references to cursor objects in PL/SQL. These cursor objects are valid SQL statements that can be iterated upon to get a set of values. These REF cursors are converted into `OracleResultSet` objects in the Java world.

You can execute these procedures, get the `OracleResultSet` object, and then send that to the `OracleXMLQuery` object to get the desired XML.

Consider the following PL/SQL function that defines a REF cursor and returns it:

```
CREATE OR REPLACE package body testRef is

    function testRefCur RETURN empREF is
        a empREF;
    begin
        OPEN a FOR select * from scott.emp;
        return a;
    end;
end;
/
```

Every time this function is called, it opens a cursor object for the query, `select * from emp` and returns that cursor instance. To convert this to XML, you can do the following:

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.xml.sql.query.OracleXMLQuery;
import java.io.*;
public class REFCURtest
{
    public static void main(String[] argv)
        throws SQLException
    {
        String str;
        Connection conn = getConnection("scott","tiger"); // create connection
```

```

// Create a ResultSet object by calling the PL/SQL function
CallableStatement stmt =
    conn.prepareCall("begin ? := testRef.testRefCur(); end;");

stmt.registerOutParameter(1,OracleTypes.CURSOR); // set the define type

stmt.execute(); // Execute the statement.
ResultSet rset = (ResultSet)stmt.getObject(1); // Get the ResultSet

OracleXMLQuery qry = new OracleXMLQuery(conn,rset); // prepare Query class
qry.setRaiseNoRowsException(true);
qry.setRaiseException(true);
qry.keepCursorState(true); // set options (keep the cursor alive..
while ((str = qry.getXMLString())!= null)
    System.out.println(str);

qry.close(); // close the query..!

// Note since we supplied the statement and resultset, closing the
// OracleXMLQuery instance will not close these. We would need to
// explicitly close this ourselves..!
stmt.close();
conn.close();
}
// Get the connection given the user name and password..!
private static Connection getConnection(String user, String passwd)
    throws SQLException
{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
    return conn;
}
}

```

To apply the stylesheet, on the other hand, use the `applyStylesheet()` command. This forces the stylesheet to be applied before generating the output.

Raising No Rows Exception

When there are no rows to process, XSU simply returns a null string. However, it might be desirable to get an exception every time there are no more rows present, so that the application can process this through exception handlers. When the

`setRaiseNoRowsException()` is set, then whenever there are no rows to generate for the output XSU raises an `oracle.xml.sql.OracleXMLSQLNoRowsException`. This is a run time exception and need not be caught unless needed.

XSU Generating XML Example 6: No Rows Exception (Java)

The following code extends the previous examples to use the exception instead of checking for null strings:

```
public class pageTest {
    .... // rest of the class definitions....

    public void main(String[] argv)
    {
        pageTest test = new pageTest("select * from emp");

        test.query.setRaiseNoRowsException(true); // ask it to generate
exceptions
        try
        {
            while(true)
                System.out.println(test.nextPage());
        }
        catch(oracle.xml.sql.OracleXMLNoRowsException)
        {
            System.out.println(" END OF OUTPUT ");
            test.close();
        }
    }
}
```

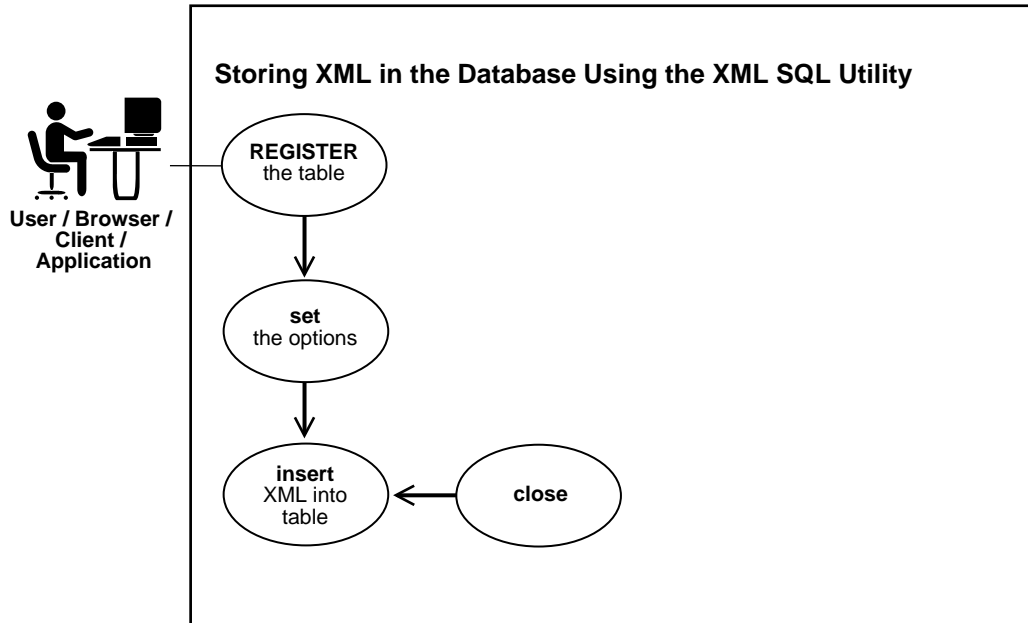
Note: Notice how the condition to check the termination changed from checking for the result to be NULL to an exception handler.

Storing XML Back in the Database Using XSU OracleXMLSave

Now that you have seen how queries can be converted to XML, observe how you can put the XML back into the tables or views using XSU. The class `oracle.xml.sql.dml.OracleXMLSave` provides this functionality. It has methods to insert XML into tables, update existing tables with the XML document, and delete rows from the table based on XML element values.

In all these cases the given XML document is parsed, and the elements are examined to match tag names to column names in the target table or view. The elements are converted to the SQL types and then bound to the appropriate statement. The process for storing XML using XSU is shown in [Figure 7-6](#).

Figure 7-6 Storing XML in the Database Using XML SQL Utility



Consider an XML document that contains a list of ROW elements, each of which constitutes a separate DML operation, namely, `insert`, `update`, or `delete` on the table or view.

Insert Processing Using XSU (Java API)

To insert a document into a table or view, simply supply the table or the view name and then the document. XSU parses the document (if a string is given) and then creates an `INSERT` statement into which it binds all the values. By default, XSU inserts values into all the columns of the table or view and an absent element is treated as a `NULL` value. The following example shows you how the XML document generated from the `emp` table, can be stored in the table with relative ease.

XSU Inserting XML Example 7: Inserting XML Values into All Columns (Java)

This example inserts XML values into all columns:

```
// This program takes as an argument the file name, or a url to
// a properly formatted XML document and inserts it into the SCOTT.EMP table.
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testInsert
{
    public static void main(String argv[])
        throws SQLException
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:oci8:@", "scott", "tiger");

        OracleXMLSave sav = new OracleXMLSave(conn, "emp");
        sav.insertXML(sav.getUrl(argv[0]));
        sav.close();
    }
}
```

An `INSERT` statement of the form:

```
insert into scott.emp (EMPNO, ENAME, JOB, MGR, SAL, DEPTNO) VALUES(?,?,?,?,?,?);
```

is generated, and the element tags in the input XML document matching the column names are matched and their values bound.

If you store the following XML document:

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
```

```

<JOB>CLERK</JOB>
<MGR>7902</MGR>
<HIREDATE>12/17/1980 0:0:0</HIREDATE>
<SAL>800</SAL>
<DEPTNO>20</DEPTNO>
</ROW>
<!-- additional rows ... -->
</ROWSET>

```

to a file and specify the file to the program described above, you would end up with a new row in the emp table containing the values (7369, Smith, CLERK, 7902, 12/17/1980, 800, 20). Any element absent inside the row element is taken as a null value.

XSU Inserting XML Example 8: Inserting XML Values into Only Certain Columns

In certain cases, you may not want to insert values into *all* columns. This may be true when the group of values that you are getting is not the complete set and you need triggers or default values to be used for the rest of the columns. The code below shows how this can be done.

Assume that you are getting the values only for the employee number, name, and job and that the salary, manager, department number, and hire date fields are filled in automatically. First create a list of column names that you want the insert to work on and then pass it to the `OracleXMLSave` instance.

```

import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testInsert
{
    public static void main(String argv[])
        throws SQLException
    {
        Connection conn = getConnection("scott","tiger");
        OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

        String [] colNames = new String[5];
        colNames[0] = "EMPNO";
        colNames[1] = "ENAME";
        colNames[2] = "JOB";

        sav.setUpdateColumnList(colNames); // set the columns to update..!

        // Assume that the user passes in this document as the first argument!
    }
}

```

```
sav.insertXML(argv[0]);
sav.close();
}
// Get the connection given the user name and password..!
private static Connection getConnection(String user, String passwd)
    throws SQLException
{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn =
        DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
    return conn;
}
}
```

An insert statement of the form:

```
insert into scott.emp (EMPNO, ENAME, JOB) VALUES (?, ?, ?);
```

is generated. Note that, in the above example, if the inserted document contains values for the other columns (JOB, HIREDATE, and so on), those are ignored. Also an insert is performed for each ROW element that is present in the input. These inserts are batched by default.

Update Processing Using XSU (Java API)

Now that you know how to insert values into the table from XML documents, see how you can update only *certain* values. In an XML document, to update the salary of an employee and the department that they work in:

```
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <SAL>1800</SAL>
    <DEPTNO>30</DEPTNO>
  </ROW>
  <ROW>
    <EMPNO>2290</EMPNO>
    <SAL>2000</SAL>
    <HIREDATE>12/31/1992</HIREDATE>
  <!-- additional rows ... -->
</ROWSET>
```

You can use the XSU to update the values. For updates, you must supply XSU with the list of key column names. These form part of the WHERE clause in the UPDATE

statement. In the emp table shown above, employee number (EMPNO) column forms the key. Use this for updates.

XSU Updating XML Example 9: Updating a Table Using the keyColumns (Java)

This example updates table, emp, using keyColumns:

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testUpdate
{
    public static void main(String argv[])
        throws SQLException
    {
        Connection conn = getConnection("scott","tiger");
        OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

        String [] keyColNames = new String[1];
        keyColNames[0] = "EMPNO";
        sav.setKeyColumnList(keyColNames);

        // Assume that the user passes in this document as the first argument!
        sav.updateXML(argv[0]);
        sav.close();
    }
    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
        throws SQLException
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
        return conn;
    }
}
```

In this example, two UPDATE statements are generated. For the first ROW element, you generate an UPDATE statement to update the SAL and JOB fields as follows:

```
update scott.emp SET SAL = 1800 and DEPTNO = 30 WHERE EMPNO = 7369;
```

For the second ROW element:

```
update scott.emp SET SAL = 2000 and HIREDATE = 12/31/1992 WHERE EMPNO = 2290;
```

XSU Updating XML Example 10: Updating a Specified List of Columns (Java)

You may want to specify a *list* of columns to update. This would speed up the processing since the same `UPDATE` statement can be used for all the `ROW` elements. Also you can ignore other tags in the XML document.

Note: When you specify a list of columns to update, an element corresponding to one of the update columns, if absent, will be treated as `NULL`.

If you know that all the elements to be updated are the same for all the `ROW` elements in the XML document, you can use the `setUpdateColumnNames()` function to set the list of columns to update.

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testUpdate
{
    public static void main(String argv[])
        throws SQLException
    {
        Connection conn = getConnection("scott","tiger");
        OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

        String [] keyColNames = new String[1];
        keyColNames[0] = "EMPNO";
        sav.setKeyColumnList(keyColNames);

        // you create the list of columns to update..!
        // Note that if you do not supply this, then for each ROW element in the
        // XML document, you would generate a new update statement to update all
        // the tag values (other than the key columns)present in that element.
        String[] updateColNames = new String[2];
        updateColNames[0] = "SAL";
        updateColNames[1] = "JOB";
        sav.setUpdateColumnList(updateColNames); // set the columns to update..!

        // Assume that the user passes in this document as the first argument!
        sav.updateXML(argv[0]);
        sav.close();
    }
    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
```

```

        throws SQLException
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
        return conn;
    }
}

```

Delete Processing Using XSU (Java API)

When deleting from XML documents, you can set the list of key columns. These columns are used in the `WHERE` clause of the `DELETE` statement. If the key column names are not supplied, then a new `DELETE` statement is created for each `ROW` element of the XML document, where the list of columns in the `WHERE` clause of the `DELETE` statement will match those in the `ROW` element.

XSU Deleting XML Example 11: Deleting Operations Per ROW (Java)

Consider this delete example:

```

import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testDelete
{
    public static void main(String argv[])
        throws SQLException
    {
        Connection conn = getConnection("scott","tiger");
        OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

        // Assume that the user passes in this document as the first argument!
        sav.deleteXML(argv[0]);
        sav.close();
    }
    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
        throws SQLException
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
        return conn;
    }
}

```

```
}
```

Using the same XML document shown previously for the update example, you would end up with two DELETE statements:

```
DELETE FROM scott.emp WHERE empno=7369 and sal=1800 and deptno=30;
DELETE FROM scott.emp WHERE empno=2200 and sal=2000 and hiredate=12/31/1992;
```

The DELETE statements were formed based on the tag names present in each ROW element in the XML document.

XSU Deleting XML Example 12: Deleting Specified Key Values (Java)

If instead, you want the DELETE statement to only use the key values as predicates, you can use the `setKeyColumn` function to set this.

```
import java.sql.*;
import oracle.xml.sql.dml.OracleXMLSave;
public class testDelete
{
    public static void main(String argv[])
        throws SQLException
    {
        Connection conn = getConnection("scott","tiger");
        OracleXMLSave sav = new OracleXMLSave(conn, "scott.emp");

        String [] keyColNames = new String[1];
        keyColNames[0] = "EMPNO";
        sav.setKeyColumnList(keyColNames);

        // Assume that the user passes in this document as the first argument!
        sav.deleteXML(argv[0]);
        sav.close();
    }
    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
        throws SQLException
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
        return conn;
    }
}
```

Here is a single DELETE statement of the form:

```
DELETE FROM scott.emp WHERE EMPNO=?
```

This is generated and used for all ROW elements in the document.

XSU PL/SQL API

The XML SQL Utility PL/SQL API reflects the Java API in the generation and storage of XML documents from and to a database. `DBMS_XMLQuery` and `DBMS_XMLSave` are the two packages that reflect the functions in the Java classes - `OracleXMLQuery` and `OracleXMLSave`. Both of these packages have a context handle associated with them. Create a context by calling one of the constructor-like functions to get the handle and then use the handle in all subsequent calls.

Generating XML with `DBMS_XMLQuery()`

Generating XML results in a CLOB that contains the XML document. To use `DBMS_XMLQuery` and the XSU generation engine, follow these steps:

1. Create a context handle by calling the `DBMS_XMLQuery.getCtx` function and supplying it the query, either as a CLOB or a VARCHAR2.
2. Bind possible values to the query using the `DBMS_XMLQuery.bind` function. The binds work by binding a name to the position. For example, the query can be `select * from emp where empno = :EMPNO_VAR`. Here you are binding the value for the `EMPNO_VAR` using the `setBindValue` function.
3. Set optional arguments like the ROW tag name, the ROWSET tag name, or the number of rows to fetch, and so on.
4. Fetch the XML as a CLOB using the `getXML()` functions. `getXML()` can be called to generate the XML with or without a DTD.
5. Close the context.

Here are some examples that use the `DBMS_XMLQuery` PL/SQL package.

XSU Generating XML Example 13: Generating XML From Simple Queries (PL/SQL)

In this example, you select rows from table `emp`, and obtain an XML document as a CLOB. First get the context handle by passing in a query and then call the `getXMLClob` routine to get the CLOB value. The document is in the same encoding as the database character set.

```
declare
    queryCtx DBMS_XMLQuery.ctxType;
    result CLOB;
begin
    -- set up the query context...!
    queryCtx := DBMS_XMLQuery.newContext('select * from emp');

    -- get the result..!
    result := DBMS_XMLQuery.getXML(queryCtx);
    -- Now you can use the result to put it in tables/send as messages..
    printClobOut(result);
    DBMS_XMLQuery.closeContext(queryCtx); -- you must close the query handle..
end;
/
```

XSU Generating XML Example 13a: Printing CLOB to Output Buffer

`printClobOut()` is a simple procedure that prints the CLOB to the output buffer. If you run this PL/SQL code in SQL*Plus, the result of the CLOB is printed to screen. Set the `serveroutput` to on in order to see the results.

```
/CREATE OR REPLACE PROCEDURE printClobOut(result IN OUT NOCOPY CLOB) is
xmlstr varchar2(32767);
line varchar2(2000);
begin
    xmlstr := dbms_lob.SUBSTR(result,32767);
    loop
        exit when xmlstr is null;
        line := substr(xmlstr,1,instr(xmlstr,chr(10))-1);
        dbms_output.put_line(' '||line);
        xmlstr := substr(xmlstr,instr(xmlstr,chr(10))+1);
    end loop;
end;
/
```

XSU Generating XML Example 14: Changing ROW and ROWSET Tags (PL/SQL)

With the XSU PL/SQL API you can also change the ROW and the ROWSET tag names. These are the default names placed around each row of the result, and round the whole document, respectively. The procedures, `setRowTagName` and `setRowSetTagName` accomplish this as shown in the following example:

```
--Setting the ROW tag names
```

```

declare
    queryCtx DBMS_XMLQuery.ctxType;
    result CLOB;
begin
    -- set the query context.
    queryCtx := DBMS_XMLQuery.newContext('select * from emp');

    DBMS_XMLQuery.setRowTag(queryCtx,'EMP'); -- sets the row tag name
    DBMS_XMLQuery.setRowSetTag(queryCtx,'EMPSET'); -- sets rowset tag name

    result := DBMS_XMLQuery.getXML(queryCtx); -- get the result

    printClobOut(result); -- print the result..!
    DBMS_XMLQuery.closeContext(queryCtx); -- close the query handle;
end;
/

```

The resulting XML document has an `EMPSET` document element. Each row is separated using the `EMP` tag.

XSU Generating XML Example 15: Using `setMaxRows()` and `setSkipRows()`

The results from the query generation can be paginated by using:

- `setMaxRows` function. This sets the maximum number of rows to be converted to XML. This is relative to the current row position from which the last result was generated.
- `setSkipRows` function. This specifies the number of rows to skip before converting the row values to XML.

For example, to skip the first 3 rows of the `emp` table and then print out the rest of the rows 10 at a time, you can set the `skipRows` to 3 for the first batch of 10 rows and then set `skipRows` to 0 for the rest of the batches.

As in the case of XML SQL Utility's Java API, call the `keepObjectOpen()` function to ensure that the state is maintained between fetches. The default behavior is to close the state after a fetch. For multiple fetches, you must determine when there are no more rows to fetch. This can be done by setting the `setRaiseNoRowsException()`. This causes an exception to be raised if no rows are written to the CLOB. This can be caught and used as the termination condition.

```
-- Pagination of results
```

```
declare
```

```
    queryCtx DBMS_XMLQuery.ctxType;
    result CLOB;
begin

    -- set up the query context...!
    queryCtx := DBMS_XMLQuery.newContext('select * from emp');

    DBMS_XMLQuery.setSkipRows(queryCtx,3); -- set the number of rows to skip
    DBMS_XMLQuery.setMaxRows(queryCtx,10); -- set the max number of rows per fetch

    result := DBMS_XMLQuery.getXML(queryCtx); -- get the first result..!

    printClobOut(result); -- print the result out.. This is you own routine..!
    DBMS_XMLQuery.setSkipRows(queryCtx,0); -- from now don't skip any more rows..!

    DBMS_XMLQuery.setRaiseNoRowsException(queryCtx,true);
                                                -- raise no rows exception..!

begin
    loop -- loop forever..!
        result := DBMS_XMLQuery.getXML(queryCtx); -- get the next batch
        printClobOut(result); -- print the next batch of 10 rows..!
    end loop;
exception
    when others then
        -- dbms_output.put_line(sqlerrm);
        null; -- termination condition, nothing to do;
end;
DBMS_XMLQuery.closeContext(queryCtx); -- close the handle..!
end;
/
```

Setting Stylesheets in XSU (PL/SQL)

The XSU PL/SQL API provides the ability to set stylesheets on the generated XML documents as follows:

- Set the stylesheet header in the result XML. To do this, use `setStyleSheetHeader()` procedure, to set the stylesheet header in the result. This simply adds the XML processing instruction to include the stylesheet.
- Apply a stylesheet to the result XML document, before generation. This method is a huge performance win since otherwise the XML document has to be generated as a CLOB, sent to the parser again, and then have the stylesheet applied. XSU generates a DOM document, calls the parser, applies the

stylesheet and then generates the result. To apply the stylesheet to the resulting XML document, use the `useStyleSheet()` procedure. This uses the stylesheet to generate the result.

Binding Values in XSU (PL/SQL)

The XSU PL/SQL API provides the ability to bind values to the SQL statement. The SQL statement can contain named bind variables. The variables must be prefixed with a colon (:) to declare that they are bind variables. To use the bind variable follow these steps:

1. **Initialize the query context with the query containing the bind variables.** For example, the following statement registers a query to select the rows from the emp table with the where clause containing the bind variables :EMPNO and :ENAME . You will bind the values for employee number and employee name later.

```
queryCtx = DBMS_XMLQuery.getCtx('select * from emp where empno = :EMPNO and
ename = :ENAME');
```

2. **Set the list of bind values.** The `clearBindValues()` clears all the bind variables set. The `setBindValue()` sets a single bind variable with a string value. For example, you will set the empno and ename values as shown below:-

```
DBMS_XMLQuery.clearBindValues(queryCtx);
DBMS_XMLQuery.setBindValue(queryCtx, 'EMPNO', 20);
DBMS_XMLQuery.setBindValue(queryCtx, 'ENAME', 'John');
```

3. **Fetch the results.** This will apply the bind values to the statement and then get the result corresponding to the predicate empno = 20 and ename = 'John'.

```
DBMS_XMLQuery.getXMLClob(queryCtx);
```

4. **Re-bind values if necessary.** For example to change the ENAME alone to scott and re-execute the query,

```
DBMS_XMLQuery.setBindValue(queryCtx, 'ENAME', 'Scott');
```

The rebinding of ENAME will now use Scott instead of John.

XSU Generating XML Example 15a: Binding Values to the SQL Statement

The following example illustrates the use of bind variables in the SQL statement:

```
declare
```

```
    queryCtx DBMS_XMLQuery.ctxType;
    result CLOB;
begin

    queryCtx := DBMS_XMLQuery.newContext(
        'select * from emp where empno = :EMPNO and ename = :ENAME');

    DBMS_XMLQuery.clearBindValues(queryCtx);
    DBMS_XMLQuery.setBindValue(queryCtx, 'EMPNO', 7566);
    DBMS_XMLQuery.setBindValue(queryCtx, 'ENAME', 'JONES');

    result := DBMS_XMLQuery.getXML(queryCtx);

    --printClobOut(result);

    DBMS_XMLQuery.setBindValue(queryCtx, 'ENAME', 'Scott');

    result := DBMS_XMLQuery.getXML(queryCtx);

    --printClobOut(result);
end;
/
```

Storing XML in the Database Using DBMS_XMLSave

To use DBMS_XMLSave() and XML SQL Utility storage engine, follow these steps:

1. Create a context handle by calling the DBMS_XMLSave.getCtx function and supplying it the table name to use for the DML operations.
2. **For inserts.** You can set the list of columns to insert into using the setUpdateColNames function. The default is to insert values into all the columns.

For updates. The list of key columns must be supplied. Optionally the list of columns to update may also be supplied. In this case, the tags in the XML document matching the key column names will be used in the WHERE clause of the update statement and the tags matching the update column list will be used in the SET clause of the update statement.

For deletes. The default is to create a WHERE clause to match all the tag values present in each ROW element of the document supplied. To override this behavior you can set the list of key columns. In this case only those tag values whose tag names match these columns will be used to identify the rows to delete (in effect used in the WHERE clause of the delete statement).

3. Supply an XML document to the `insertXML`, `updateXML`, or `deleteXML` functions to insert, update and delete respectively.
4. You can repeat the last operation any number of times.
5. Close the context.

Use the same examples as for the Java case, `OracleXMLSave` class examples.

Insert Processing Using XSU (PL/SQL API)

To insert a document into a table or view, simply supply the table or the view name and then the XML document. XSU parses the XML document (if a string is given) and then creates an INSERT statement, into which it binds all the values. By default, XSU inserts values into all the columns of the table or view and an absent element is treated as a NULL value.

The following code shows how the document generated from the emp table can be put back into it with relative ease.

XSU Inserting XML Example 16: Inserting Values into All Columns (PL/SQL)

This example creates a procedure, `insProc`, which takes in:

- An XML document as a CLOB
- A table name to put the document into

and then inserts the XML document into the table:

```
create or replace procedure insProc(xmlDoc IN CLOB, tableName IN VARCHAR2) is
  insCtx DBMS_XMLSave.ctxType;
  rows number;
begin
  insCtx := DBMS_XMLSave.newContext(tableName); -- get the context handle
  rows := DBMS_XMLSave.insertXML(insCtx,xmlDoc); -- this inserts the document
  DBMS_XMLSave.closeContext(insCtx);           -- this closes the handle
end;
/
```

This procedure can now be called with any XML document and a table name. For example, a call of the form:

```
insProc(xmlDocument, 'scott.emp');
```

generates an INSERT statement of the form:

```
insert into scott.emp (EMPNO, ENAME, JOB, MGR, SAL, DEPTNO) VALUES(?,?,?,?);
```

and the element tags in the input XML document matching the column names will be matched and their values bound. For the code snippet shown above, if you send it the following XML document:

```
<?xml version='1.0'?>
<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <ENAME>Smith</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
    <SAL>800</SAL>
    <DEPTNO>20</DEPTNO>
  </ROW>
  <!-- additional rows ... -->
</ROWSET>
```

you would have a new row in the emp table containing the values (7369, Smith, CLERK, 7902, 12/17/1980,800,20). Any element absent inside the row element would be considered a null value.

XSU Inserting XML Example 17: Inserting Values into Certain Columns (PL/SQL)

In certain cases, you may not want to insert values into all columns. This might be true when the values that you are getting is not the complete set and you need triggers or default values to be used for the rest of the columns. The code below shows how this can be done.

Assume that you are getting the values only for the employee number, name, and job, and that the salary, manager, department number and hiredate fields are filled in automatically. You create a list of column names that you want the insert to work on and then pass it to the DBMS_XMLSave procedure. The setting of these values can be done by calling `setUpdateColumnName()` procedure repeatedly, passing in a column name to update every time. The column name settings can be cleared using `clearUpdateColumnNames()`.

```
create or replace procedure testInsert( xmlDoc IN clob) is
  insCtx DBMS_XMLSave.ctxType;
  doc clob;
  rows number;
begin
```

```

insCtx := DBMS_XMLSave.newContext('scott.emp'); -- get the save context..!

DBMS_XMLSave.clearUpdateColumnList(insCtx); -- clear the update settings

-- set the columns to be updated as a list of values..
DBMS_XMLSave.setUpdateColumn(insCtx, 'EMPNO' );
DBMS_XMLSave.setUpdateColumn(insCtx, 'ENAME' );
DBMS_XMLSave.setUpdatecolumn(insCtx, 'JOB' );

-- Now insert the doc. This will only insert into EMPNO,ENAME and JOB columns
rows := DBMS_XMLSave.insertXML(insCtx, xmlDoc);
DBMS_XMLSave.closeContext(insCtx);

end;
/

```

If you call the procedure passing in a CLOB as a document, an INSERT statement of the form:

```
insert into scott.emp (EMPNO, ENAME, JOB) VALUES (?, ?, ?);
```

is generated. Note that in the above example, if the inserted document contains values for the other columns (JOB, HIREDATE, and so on), those are ignored.

Also an insert is performed for each ROW element that is present in the input. These inserts are batched by default.

Update Processing Using XSU (PL/SQL API)

Now that you know how to insert values into the table from XML documents, let us see how to update only certain values. If you get an XML document to update the salary of an employee and also the department that she works in:

```

<ROWSET>
  <ROW num="1">
    <EMPNO>7369</EMPNO>
    <SAL>1800</SAL>
    <DEPTNO>30</DEPTNO>
  </ROW>
  <ROW>
    <EMPNO>2290</EMPNO>
    <SAL>2000</SAL>
    <HIREDATE>12/31/1992</HIREDATE>
  <!-- additional rows ... -->
</ROWSET>

```

you can call the update processing to update the values. In the case of update, you need to supply XSU with the list of key column names. These form part of the where clause in the update statement. In the emp table shown above, the employee number (EMPNO) column forms the key and you use that for updates.

XSU Updating XML Example 18: Updating an XML Document Using keyColumns

```
.....
create or replace procedure testUpdate ( xmlDoc IN clob) is
  updCtx DBMS_XMLSave.ctxType;
  rows number;
begin

  updCtx := DBMS_XMLSave.newContext('scott.emp'); -- get the context
  DBMS_XMLSave.clearUpdateColumnList(updCtx); -- clear the update settings..

  DBMS_XMLSave.setKeyColumn(updCtx,'EMPNO'); -- set EMPNO as key column
  rows := DBMS_XMLSave.updateXML(updCtx,xmlDoc); -- update the table.
  DBMS_XMLSave.closeContext(updCtx);          -- close the context..!

end;
/
```

In this example, when the procedure is executed with a CLOB value that contains the document described above, two update statements would be generated. For the first ROW element, you would generate an UPDATE statement to update the SAL and JOB fields as shown below:-

```
update scott.emp SET SAL = 1800 and DEPTNO = 30 WHERE EMPNO = 7369;
```

and for the second ROW element,

```
update scott.emp SET SAL = 2000 and HIREDATE = 12/31/1992 WHERE EMPNO = 2290;
```

XSU Updating XML Example 19: Specifying a List of Columns to Update (PL/SQL)

You may want to specify the list of columns to update. This would speed up the processing since the same update statement can be used for all the ROW elements. Also you can ignore other tags which occur in the document. Note that when you specify a list of columns to update, an element corresponding to one of the update columns, if absent, will be treated as NULL.

If you know that all the elements to be updated are the same for all the ROW elements in the XML document, then you can use the `setUpdateColumnName()` procedure to set the column name to update.

```

create or replace procedure testUpdate(xmlDoc IN CLOB) is
  updCtx DBMS_XMLSave.ctxType;
  rows number;
begin

  updCtx := DBMS_XMLSave.newContext('scott.emp');
  DBMS_XMLSave.setKeyColumn(updCtx,'EMPNO'); -- set EMPNO as key column

  -- set list of columnst to update.
  DBMS_XMLSave.setUpdateColumn(updCtx,'SAL');
  DBMS_XMLSave.setUpdateColumn(updCtx,'JOB');

  rows := DBMS_XMLSave.updateXML(updCtx,xmlDoc); -- update the XML document..!
  DBMS_XMLSave.closeContext(updCtx); -- close the handle

end;
/

```

Delete Processing Usingh XSU (PL/SQL API)

For deletes, you can set the list of key columns. These columns will be put as part of the WHERE clause of the DELETE statement. If the key column names are not supplied, then a new DELETE statement will be created for each ROW element of the XML document where the list of columns in the WHERE clause of the DELETE will match those in the ROW element.

XSU Deleting XML Example 20: Deleting Operations per ROW (PL/SQL)

Consider the delete example shown here:

```

create or replace procedure testDelete(xmlDoc IN clob) is
  delCtx DBMS_XMLSave.ctxType;
  rows number;
begin

  delCtx := DBMS_XMLSave.newContext('scott.emp');
  DBMS_XMLSave.setKeyColumn(delCtx,'EMPNO');

  rows := DBMS_XMLSave.deleteXML(delCtx,xmlDoc);
  DBMS_XMLSave.closeContext(delCtx);

```

```
end;  
/
```

If you use the same XML document shown for the update example, you would end up with two DELETE statements,

```
DELETE FROM scott.emp WHERE empno=7369 and sal=1800 and deptno=30;  
DELETE FROM scott.emp WHERE empno=2200 and sal=2000 and hiredate=12/31/1992;
```

The DELETE statements were formed based on the tag names present in each ROW element in the XML document.

XSU Example 21: Deleting by Specifying the Key Values (PL/SQL)

If instead you want the delete to only use the key values as predicates, you can use the `setKeyColumn` function to set this.

```
create or replace package testDML AS  
    saveCtx DBMS_XMLSave.ctxType := null;    -- a single static variable  
  
    procedure insertXML(xmlDoc in clob);  
    procedure updateXML(xmlDoc in clob);  
    procedure deleteXML(xmlDoc in clob);  
  
end;  
/  
  
create or replace package body testDML AS  
  
    rows number;  
  
    procedure insertXML(xmlDoc in clob) is  
    begin  
        rows := DBMS_XMLSave.insertXML(saveCtx,xmlDoc);  
    end;  
  
    procedure updateXML(xmlDoc in clob) is  
    begin  
        rows := DBMS_XMLSave.updateXML(saveCtx,xmlDoc);  
    end;  
  
    procedure deleteXML(xmlDoc in clob) is  
    begin  
        rows := DBMS_XMLSave.deleteXML(saveCtx,xmlDoc);  
    end;
```



```

begin
    saveCtx := DBMS_XMLSave.newContext('scott.emp'); -- create the context once..!
    DBMS_XMLSave.setKeyColumn(saveCtx, 'EMPNO');      -- set the key column name.
end;
/

```

Here a single delete statement of the form,

```
DELETE FROM scott.emp WHERE EMPNO=?
```

will be generated and used for all ROW elements in the document.

XSU Deleting XML Example 22: ReUsing the Context Handle (PL/SQL)

In all the three cases described above, insert, update, and delete, the same context handle can be used to do more than one operation. That is, you can perform more than one insert using the same context provided all of those inserts are going to the same table that was specified when creating the save context. The context can also be used to mix updates, deletes, and inserts.

For example, the following code shows how one can use the same context and settings to insert, delete, or update values depending on the user's input.

The example uses a PL/SQL supplied package static variable to store the context so that the same context can be used for all the function calls.

```

create or replace package testDML AS
    saveCtx DBMS_XMLSave.ctxType := null; -- a single static variable

    procedure insert(xmlDoc in clob);
    procedure update(xmlDoc in clob);
    procedure delete(xmlDoc in clob);

end;
/

create or replace package body testDML AS

    procedure insert(xmlDoc in clob) is
    begin
        DBMS_XMLSave.insertXML(saveCtx, xmlDoc);
    end;

    procedure update(xmlDoc in clob) is

```

```
begin
    DBMS_XMLSave.updateXML(saveCtx, xmlDoc);
end;

procedure delete(xmlDoc in clob) is
begin
    DBMS_XMLSave.deleteXML(saveCtx, xmlDoc);
end;

begin
    saveCtx := DBMS_XMLSave.newContext('scott.emp'); -- create the context
once..!
    DBMS_XMLSave.setKeyColumn(saveCtx, 'EMPNO'); -- set the key column name.
end;
end;
/
```

In the above package, you create a context once for the whole package (thus the session) and then reuse the same context for performing inserts, updates and deletes.

Note: The key column EMPNO would be used both for updates and deletes as a way of identifying the row.

Users of this package can now call any of the three routines to update the emp table:

```
testDML.insert(xmlclob);
testDML.delete(xmlclob);
testDML.update(xmlclob);
```

All of these calls would use the same context. This would improve the performance of these operations, particularly if these operations are performed frequently.

Advanced XSU Usage Techniques

XSU Exception Handling in Java

OracleXMLSQLException class

XSU catches all exceptions that occur during processing and throws an `oracle.xml.sql.OracleXMLSQLException` which is a run time exception. The calling program thus does not have to catch this exception all the time, if the

program can still catch this exception and do the appropriate action. The exception class provides functions to get the error message and also get the parent exception, if any. For example, the program shown below, catches the run time exception and then gets the parent exception.

OracleXMLNoRowsException class

This exception is generated when the `setRaiseNoRowsException` is set in the `OracleXMLQuery` class during generation. This is a subclass of the `OracleXMLSQLException` class and can be used as an indicator of the end of row processing during generation.

```
import java.sql.*;
import oracle.xml.sql.query.OracleXMLQuery;

public class testException
{
    public static void main(String argv[])
        throws SQLException
    {
        Connection conn = getConnection("scott","tiger");

        // wrong query this will generate an exception
        OracleXMLQuery qry = new OracleXMLQuery(conn, "select * from emp where sd
= 322323");

        qry.setRaiseException(true); // ask it to raise exceptions..!

        try{
            String str = qry.getXMLString();
        }catch(oracle.xml.sql.OracleXMLSQLException e)
        {
            // Get the original exception
            Exception parent = e.getParentException();
            if (parent instanceof java.sql.SQLException)
            {
                // perform some other stuff. Here you simply print it out..
                System.out.println(" Caught SQL Exception:"+parent.getMessage());
            }
            else
                System.out.println(" Exception caught..!" +e.getMessage());
        }
    }
    // Get the connection given the user name and password..!
    private static Connection getConnection(String user, String passwd)
```

```
        throws SQLException
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn =
            DriverManager.getConnection("jdbc:oracle:oci8:@",user,passwd);
        return conn;
    }
}
```

XSU Exception Handling in PL/SQL

Here is an XSU PL/SQL exception handling example:

```
declare
    queryCtx DBMS_XMLQuery.ctxType;
    result clob;
    errorNum NUMBER;
    errorMsg VARCHAR2(200);
begin

    queryCtx := DBMS_XMLQuery.newContext('select * from emp where df = ddfd');

    -- set the raise exception to true..
    DBMS_XMLQuery.setRaiseException(queryCtx, true);
    DBMS_XMLQuery.setRaiseNoRowsException(queryCtx, true);

    -- set propagate original exception to true to get the original exception..!
    DBMS_XMLQuery.propagateOriginalException(queryCtx, true);
    result := DBMS_XMLQuery.getXML(queryCtx);

    exception
    when others then
        -- get the original exception
        DBMS_XMLQuery.getExceptionContent(queryCtx,errorNum, errorMsg);
        dbms_output.put_line(' Exception caught ' || TO_CHAR(errorNum)
            || errorMsg );
end;
/
```

Frequently Asked Questions (FAQs): XML SQL Utility (XSU)

What Schema Structure Should I Use With XSU to Store XML?

I have the following XML in my customer.xml file:

```

<ROWSET>
  <ROW num="1">
    <CUSTOMER>
      <CUSTOMERID>1044</CUSTOMERID>
      <FIRSTNAME>Paul</FIRSTNAME>
      <LASTNAME>Astoria</LASTNAME>
      <HOMEADDRESS>
        <STREET>123 Cherry Lane</STREET>
        <CITY>SF</CITY>
        <STATE>CA</STATE>
        <ZIP>94132</ZIP>
      </HOMEADDRESS>
    </CUSTOMER>
  </ROW>
</ROWSET>

```

What database schema structure should I use to store this XML with XSU?

Answer

Since your example is more than one level deep (that is, it has a nested structure), you should use an object-relational schema. The XML above will canonically map to such a schema. An appropriate database schema would be the following:

```

create type address_type as object
(
  street varchar2(40),
  city varchar2(20),
  state varchar2(10),
  zip varchar2(10)
);
/
create type customer_type as object
(
  customerid number(10),
  firstname varchar2(20),
  lastname varchar2(20),
  homeaddress address_type
);
/
create table customer_tab ( customer customer_type);

```

In the case you wanted to load `customer.xml` via the XSU into a relational schema, you could still do it by creating objects in views on top of your relational schema.

For example, you would have a relational table which would contain all the information:

```
create table cust_tab
( customerid number(10),
  firstname varchar2(20),
  lastname varchar2(20),
  state varchar2(40),
  city varchar2(20),
  state varchar2(20),
  zip varchar2(20)
);
```

Then you would create a customer view which contains a customer object on top of it, as in the following example:

```
create view customer_view as
select customer_type(customerid, firstname, lastname,
address_type(state,street,city,zip))
from cust_tab;
```

Finally, you could flatten your XML using XSLT and then insert it directly into your relational schema. However, this is the least recommended option.

Can XSU Store XML Data Across Tables?

Can XML SQL Utility store XML data across tables?

Answer

Currently XML SQL Utility (XSU) can only store to a single table. It maps a canonical representation of an XML document into any table or view. But of course there is a way to store XML with XSU across tables. One can do this using XSLT to transform any document into multiple documents and insert them separately. Another way is to define views over multiple tables (object views if needed) and then do the `inserts` into the view. If the view is inherently non-updatable (because of complex joins), then you can use `INSTEAD OF` triggers over the views to do the inserts.

Can I Use XML SQL Utility to Load XML Stored in Attributes?

I would like to use XML SQL Utility to load XML where some of the data is stored in attributes; yet, XML SQL Utility seems to ignore the XML attributes. What can I do?

Answer

Unfortunately, for now you will have to use XSLT to transform your XML document (that is, change your attributes into elements). XML SQL Utility does assume canonical mapping from XML to a database schema. This takes away a bit from the flexibility, forcing you to sometimes resort to XSLT, but at the same time, in the common case, it does not burden you with having to specify a mapping.

Is XML SQL Utility Case Sensitive? Can I Use ignoreCase?

I am trying to insert the following XML document (dual.xml):

```
<ROWSET>
  <row>
    <DUMMY>X</DUMMY>
  </row>
</ROWSET>
```

Into the table `dual` using the command line front end of the XSU, like in:

```
java OracleXML putxml -filename dual.xml dual
```

and I get the following error:

```
oracle.xml.sql.OracleXMLSQLException: No rows to modify -- the row enclosing tag
missing. Specify the correct row enclosing tag.
```

Answer

By default, XML SQL Utility is case sensitive, so it looks for the record separator tag which by default is `ROW`, yet, all it can find is `row`. Another related common mistake is to case mismatch one of the element tags. For example, if in `dual.xml` the tag `DUMMY` was actually `dummy`, then XML SQL Utility raises an error complaining that it could not find a matching column in table, `dual`. So you have two options -- use the correct case or use the `ignoreCase` feature.

Will XSU Generate Database Schema from a DTD?

Given a DTD, will XML SQL Utility generate the database schema?

Answer

No. Due to a number of shortcomings of the DTD, this functionality is not available. Once the W3C XML Schema recommendation is finalized this functionality will become feasible.

Can You Provide a Thin Driver Connect String Example for XSU?

I am using the XML SQL Utility command line front end, and I am passing a connect string but I get a TNS error back. Can you provide examples of a thin driver connect string and an OCI8 driver connect string?

Answer

An example of an JDBC thin driver connect string is:

```
jdbc:oracle:thin:<user>/<password>@<hostname>:<port number>:<DB SID>;
```

furthermore, the database must have an active TCP/IP listener. A valid OCI8 connect string would be:

```
jdbc:oracle:oci8:<user>/<password>@<hostname>
```

Does XML SQL Utility Commit After INSERT, DELETE, UPDATE?

Does XML SQL Utility commit after it is done inserting, deleting, or updating? What happens if an error occurs?

Answer

By default XML SQL Utility executes a number of insert, delete, or update statements at a time. The number of statements batch together and executed at the same time can be overridden using the `setBatchSize` feature.

Also, by default XML SQL Utility does no explicit commits. If `autocommit` is on (default for the JDBC connection), then after each batch of statement executions a commit occurs. You can override this by turning `autocommit` off and then specifying after how many statement executions should a commit occur which can be done using the `setCommitBatch` feature.

What happens if an error occurs? XSU rolls back to either the state the target table was before the particular call to XSU, or the state right after the last commit made during the current call to XSU.

Can You Explain How to Map Table Columns to XML Attributes Using XSU?

Question

Can you explain how to map table columns to XML attributes using XSU?

Answer

From XML SQL Utility release 2.1.0 you can map a particular column or a group of columns to an XML attribute instead of an XML element. To achieve this, you have to create an alias for the column name, and prepend the at sign (@) to the name of this alias. For example :

- * Create a file called select.sql with the following content :

```
SELECT empno "@EMPNO", ename, job, hiredate
FROM emp
ORDER BY empno
```

- * Call the XML SQL Utility :

```
java OracleXML getXML -user "scott/tiger" \
    -conn "jdbc:oracle:thin:@myhost:1521:ORCL" \
    -fileName "select.sql"
```

- * As a result, the XML document will look like :

```
<?xml version = '1.0'?>
<ROWSET>
  <ROW num="1" EMPNO="7369">
    <ENAME>SMITH</ENAME>
    <JOB>CLERK</JOB>
    <HIREDATE>12/17/1980 0:0:0</HIREDATE>
  </ROW>
  <ROW num="2" EMPNO="7499">
    <ENAME>ALLEN</ENAME>
    <JOB>SALESMAN</JOB>
    <HIREDATE>2/20/1981 0:0:0</HIREDATE>
  </ROW>
</ROWSET>
```

Note: All attributes must appear *before* any non-attribute.

Since the XML document is created in a streamed manner, the query :

```
SELECT ename, empno "@EMPNO", ...
```

would not generate the expected result. It is currently not possible to load XML data stored in attributes. You will still need to use an XSLT transformation to change the attributes into elements. XML SQL Utility assumes canonical mapping from XML to a database schema.

How Can I Use XMLGEN.insertXML with LOBs?

I am using the following:

- OS: SOLARIS 7
- DB: ORACLE 815

and trying to use the `insertXML` procedure from XSU. I have little experience with using LOBS. What is the problem in my script?

I have a table `lob_temp`:

```
SQL> desc lob_temp
Name Null? Type
-----
CHUNK CLOB

SQL> set long 100000
SQL> select * from lob_temp;

CHUNK
-----
<DOCID> 91739.1 </DOCID>
<SUBJECT> MTS: ORA-29855, DRG-50704, ORA-12154: on create index using
Intermedia
</SUBJECT>
<TYPE> PROBLEM </TYPE>
<CONTENT_TYPE> TEXT/PLAIN </CONTENT_TYPE>
<STATUS> PUBLISHED </STATUS>
<CREATION_DATE> 14-DEC-1999 </CREATION_DATE>
<LAST_REVISION_DATE> 05-JUN-2000 </LAST_REVISION_DATE>
<LANGUAGE> USAENG </LANGUAGE>
```

I have another table where I need to insert data from `lob_temp`:

```
SQL> desc metalink_doc
Name Null? Type
-----
```

```
DOCID VARCHAR2(10)
SUBJECT VARCHAR2(100)
TYPE VARCHAR2(20)
CONTENT_TYPE VARCHAR2(20)
STATUS VARCHAR2(20)
CREATION_DATE DATE
LAST_REVISION_DATE DATE
LANGUAGE VARCHAR2(10)
```

This is the script. It is supposed to read data from `lob_temp` and then insert the data, extracted from the XML document, to table `metalink_doc`:

```
declare
xmlstr clob := null;
amount integer := 255;
position integer := 1;
charstring varchar2(255);
finalstr varchar2(4000) := null;
ignore_case constant number := 0;
default_date_format constant varchar2(21) := 'DD-MON-YYYY';
default_rowtag constant varchar2(10) := 'MDOC_DATA';
len integer;
insrow integer;
begin
select chunk into xmlstr from lob_temp;
dbms_lob.open(xmlstr,dbms_lob.lob_readonly);
len := dbms_lob.getlength(xmlstr);
while position < len loop
dbms_lob.read(xmlstr,amount,position,charstring);
if finalstr is not null then
finalstr := finalstr||charstring;
else
finalstr := charstring;
end if;
position := position + amount;
end loop;
insrow := xmlgen.insertXML('metalink_doc',finalstr);
dbms_output.put_line(insrow);
dbms_lob.close(xmlstr);
exception
when others then
dbms_lob.close(xmlstr);
dbms_lob.freetemporary(xmlstr);
end;
/
```

This is the error received:

```
ERROR at line 1:
ORA-22275: invalid LOB locator specified
ORA-06512: at "SYS.DBMS_LOB", line 485
ORA-06512: at line 31
ORA-29532: Java call terminated by uncaught Java exception:
oracle.xml.sql.OracleXMLSQLException: Expected 'EOF'.
```

The user I am using owns both tables, and all objects created when I ran `oraclexmlsqlload.csh`.

Answer

You need to have `<ROWSET>` and `<ROW>` tags to insert XML document into a table. I modified your procedure as below. There is a problem when parsing the DATE format, hence I used VARCHAR2:

```
drop table lob_temp;
create table lob_temp (chunk clob);
insert into lob_temp values (
  <ROWSET>
  <ROW>
  <DOCID> 91739.1 </DOCID>
  <SUBJECT> MTS: ORA-29855, DRG-50704, ORA-12154: on create index using
  Intermedia </SUBJECT>
  <TYPE> PROBLEM </TYPE>
  <CONTENT_TYPE> TEXT/PLAIN </CONTENT_TYPE>
  <STATUS> PUBLISHED </STATUS>
  <CREATION_DATE> 14-DEC-1999 </CREATION_DATE>
  <LAST_REVISION_DATE> 05-JUN-2000 </LAST_REVISION_DATE>
  <LANGUAGE> USAENG </LANGUAGE>
</ROW>
</ROWSET>
');

drop table metalink_doc;
create table metalink_doc (
  DOCID VARCHAR2(10),
  SUBJECT VARCHAR2(100),
  TYPE VARCHAR2(20),
  CONTENT_TYPE VARCHAR2(20),
  STATUS VARCHAR2(20),
  CREATION_DATE VARCHAR2(50),
  LAST_REVISION_DATE varchar2(50),
```

```
LANGUAGE VARCHAR2(10)
);

create or replace procedure prtest as
xmlstr clob := null;
amount integer := 255;
position integer := 1;
charstring varchar2(255);
finalstr varchar2(4000) := null;
ignore_case constant number := 0;
default_date_format constant varchar2(21) := 'DD-MON-YYYY';
default_rowtag constant varchar2(10) := 'MDOC_DATA';
len integer;
insrow integer;
begin

select chunk into xmlstr from lob_temp;
dbms_lob.open(xmlstr,dbms_lob.lob_readonly);
len := dbms_lob.getlength(xmlstr);

while position < len loop
dbms_lob.read(xmlstr,amount,position,charstring);
if finalstr is not null then
finalstr := finalstr||charstring;
else
finalstr := charstring;
end if;
position := position + amount;
end loop;

insrow := xmlgen.insertXML('metalink_doc',finalstr);
dbms_output.put_line(insrow);

IF DBMS_LOB.ISOPEN(xmlstr) = 1 THEN
dbms_lob.close(xmlstr);
END IF;

exception
when others then
IF DBMS_LOB.ISOPEN(xmlstr)=1 THEN
dbms_lob.close(xmlstr);
END IF;
end;
/
show err
```

Response Comment

Its working! Thank you!

Searching XML Data with Oracle Text

This chapter describes the following aspects of Oracle Text (*interMedia Text/Context*):

- How to create a section group and index your XML document(s)
- How to build an XML query application with Oracle Text, to search and retrieve data from your XML document(s)

This chapter contains the following sections:

- [Introducing Oracle Text](#)
- [Assumptions Made in this Chapter's Examples](#)
- [Oracle Text Users and Roles](#)
- [Querying with the CONTAINS Operator](#)
 - [Using a Simple SELECT Statement](#)
 - [Using the Score Operator with a Label to Obtain the Relevance](#)
 - [Using the WITHIN Operator to Narrow Query Down to Document Sections](#)
 - [Using INPATH or HASPATH Operators for Query Searching](#)
- [Using Oracle Text to Search XML Documents](#)
- [Building XML Query Applications with Oracle Text](#)
 - [Querying XML Documents](#)
 - [Querying Within Attribute Sections](#)
 - [Procedure for Building a Query Application with Oracle Text](#)
 - [Step 1. Create a Preference](#)

-
- Step 2. Set the Preference's Attributes
 - Step 3. Create Your Query Syntax
 - Creating Sections in XML Documents that are Document Type Sensitive
 - Presenting the Results of Your Query
 - Case Study: Searching an Online FAQ List Using Oracle Text
 - Frequently Asked Questions (FAQs): Oracle Text

Introducing Oracle Text

Note: Oracle Text is strictly a server-based implementation.

See Also: <http://otn.oracle.com/products/text>

Oracle Text can be used to search XML documents. It extends Oracle by indexing any text or document stored in Oracle. It can also search documents in the operating system (flat files) and URLs.

Oracle Text enables the following:

- **Content-based queries**, such as, finding text and documents which contain particular words, using familiar, standard SQL.
- File-based text applications to use Oracle to **manage text and documents** in an integrated fashion with traditional relational information.
- **Concept searching** of English language documents.
- **Theme analysis** of English language documents using the theme/gist package.
- **Highlighting hit words.** With Oracle Text, you can render a document in different ways. For example, you can present documents with query terms highlighted, either the “words” of a word query or the “themes” of an ABOUT query in English. Use the CTX_DOC.MARKUP or HIGHLIGHT procedures for this.
- With Oracle Text PL/SQL packages for **document presentation and thesaurus maintenance**.

Oracle Text is packaged with the other *interMedia* products, namely, image, audio, video, and geographic location services for web content management applications.

Users can query XML data stored in the database directly, without using Oracle Text. However, Oracle Text is useful for boosting query performance.

See Also:

- *Oracle9i Text Reference*
- *Oracle9i Text Developer's Guide*
- <http://otn.oracle.com/products/text>

Accessing Oracle Text

interMedia, including Oracle Text, is a standard feature that comes with every Oracle Standard, Enterprise, and Personal edition license. It needs to be selected during installation. No special installation instructions are required.

Oracle Text is essentially a set of schema objects owned by CTXSYS. These objects are linked to the Oracle kernel. The schema objects are present when you perform an Oracle installation.

Further Oracle Text Examples

You can find more examples for Oracle Text and for creating section group indexes at the following site: <http://otn.oracle.com/products/text>

Assumptions Made in this Chapter's Examples

XML text is a VARCHAR2 or CLOB type in an Oracle database table with character semantics. Oracle Text can also deal with documents in a file system or in URLs, but we are not considering these document types in this chapter.

To simplify the examples included in this chapter we consider a subset of the Oracle Text options. In this chapter's examples, we made the following assumptions:

- All XML data here is represented using US-ASCII, a 7 bit character set.
- Issues about whether a character such as "*" is treated as white space or as part of a word are not included.
- Storage characteristics of the Oracle schema object that implement the TEXT index are not considered.
- We focus here on the SECTION GROUP parameter in the CREATE INDEX or ALTER INDEX statement. The other parameter types available for CREATE INDEX and ALTER INDEX, are DATASTORE, FILTER, LEXER, STOPLIST, and WORDLIST.

See Also: *Oracle9i Text Reference*, for more information on these parameter types.

Here is an example of using SECTION GROUP in CREATE INDEX:

```
CREATE INDEX my_index
ON my_table ( my_column )
INDEXTYPE IS ctxsys.context
PARAMETERS ( 'SECTION GROUP my_section_group' );
```

- Specifically, we focus on using `AUTO_SECTION_GROUP` and `XML_SECTION_GROUP`, and `PATH_SECTION_GROUP`.
- Tagged or marked up data. In this chapter, we focus on how to handle XML data. Oracle Text handles many other kinds of data besides XML data.

See Also: *Oracle9i Text Developer's Guide*

Oracle Text Users and Roles

With Oracle Text you can use the following users/roles:

- user `CTXSYS` to administer users
- role `CTXAPP` to create and delete Oracle Text *preferences* and use Oracle Text PL/SQL packages

User CTXSYS

This user is created at install time. Administer Oracle Text users as this user. It has the following privileges:

- Modify system-defined preferences
- Drop and modify other user preferences
- Call procedures in the `CTX_ADM` PL/SQL package to start servers and set system-parameters
- Start a `ctxsrv` server
- Query all system-defined views
- Perform all the tasks of a user with the `CTXAPP` role

Role CTXAPP

Any user can create an Oracle Text index and issue a Text query. For additional tasks, use the `CTXAPP` role. This is a system-defined role that allows you to perform the following tasks:

- Create and delete Oracle Text *preferences*
- Use Oracle Text PL/SQL packages, such as the `CTX_DDL` package

Querying with the CONTAINS Operator

Oracle Text's main purpose is to provide an implementation for the CONTAINS operator. The CONTAINS operator is used in the WHERE clause of a SELECT statement to specify the query expression for a Text query.

See Also: ["Building XML Query Applications with Oracle Text"](#).

CONTAINS Syntax

Here is the CONTAINS syntax:

```
...WHERE CONTAINS([schema.]column,text_query VARCHAR2,[label NUMBER])
```

where:

Table 8–1 CONTAINS Operator: Syntax Description

Syntax	Description
[schema.]column	Specifies the text column to be searched on. This column must have a Text index associated with it.
text_query	Specifies the query expression that defines your search in column.
label	Optionally specifies the label that identifies the score generated by the CONTAINS operator.

For each row selected, CONTAINS returns a number between 0 and 100 that indicates how relevant the document row is to the query. The number 0 means that Oracle found no matches in the row. You can obtain this score with the SCORE operator.

Note: You must use the SCORE operator with a label to obtain this number.

Using a Simple SELECT Statement

The following example illustrates how the CONTAINS operator is used in a SELECT statement:

```
SELECT id FROM my_table
WHERE
CONTAINS (my_column, 'receipts') > 0
```

The 'receipts' parameter of the CONTAINS function is called the "Text Query Expression".

Note: The SQL statement with the CONTAINS function requires a text index in order to run.

Using the Score Operator with a Label to Obtain the Relevance

The following example searches for all documents in the text column that contain the word Oracle. The score for each row is selected with the SCORE operator using a label of 1:

```
SELECT SCORE(1), title from newsindex
       WHERE CONTAINS(text, 'oracle', 1) > 0 ORDER BY SCORE(1) DESC;
```

The CONTAINS operator must always be followed by the > 0 syntax. This specifies that the score value calculated by the CONTAINS operator must be greater than zero for the row selected.

When the SCORE operator is called, such as in a SELECT clause, the operator must reference the label value as shown in the example.

Using the WITHIN Operator to Narrow Query Down to Document Sections

When documents have internal structure such as in HTML and XML, you can define document sections using embedded tags before you index. This enables you to query within the sections using the WITHIN operator.

Note: This is only true for XML_SECTION_GROUP, but not true for AUTO_ or PATH_SECTION_GROUP.

You define sections as part of a section group. Use the WITHIN operator to narrow queries down into document sections. Document sections can be any of the following:

- Zone sections
- Field sections
- Attribute sections

- Special sections (sentence or paragraph)

WITHIN Syntax for Section Querying

Here is the WITHIN syntax for querying sections:

```
expression WITHIN section
```

This searches for *expression* within a section. If you are using XML_SECTION_GROUP the following restrictions apply to the pre-defined zone, field, or attribute section:

- If section is a **zone**, expression can contain one or more WITHIN operators (nested WITHIN) whose section is a zone or special section.
- If section is a **field** or **attribute** section, expression cannot contain another WITHIN operator.

You can combine and nest WITHIN clauses. For finer grained searches of XML sections, you can use WITHIN clauses inside CONTAINS select statements.

WITHIN Operator Limitations

The WITHIN operator has the following limitations:

- You cannot embed the WITHIN clause in a phrase. For example, you cannot write: term1 WITHIN section term2
- You cannot combine WITHIN with expansion operators, such as \$! and *.
- Since WITHIN is a reserved word, you must escape the word with braces to search on it.

See Also: ■ *Oracle9i Text Reference*

You can query within attribute sections when you index with either XML_SECTION_GROUP, AUTO_SECTION_GROUP, or PATH_SECTION_GROUP your section group type.

Consider the following XML document:

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

XML_SECTION_GROUP

If you use XML_SECTION_GROUP, you can name attribute sections anything with CTX_DDL.ADD_ATTR_SECTION.

To define section, `title@book`, as the attribute section `booktitle`, you can use either of the following methods:

- ***CTX_DLL.ADD_ATTR_SECTION procedure.*** The syntax for this is:

```
CTX_DLL.ADD_ATTR_SECTION(
    group_name    in    varchar2,
    section_name  in    varchar2,
    tag           in    varchar2);
```

To define the title attribute as an attribute section, create an `XML_SECTION_GROUP` and define the attribute section as follows:

```
EXEC ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');
```

When you define the `TITLE` attribute section as such and index the document set, you can query the XML attribute text as follows:

```
'Cities within booktitle'
```

- ***Dynamically, after indexing, using the ALTER INDEX statement.*** The syntax for `ALTER INDEX` is:

```
ALTER INDEX [schema.]index REBUILD [ONLINE] [PARAMETERS (paramstring)];
```

where

```
paramstring = 'replace [datastore datastore_pref]
               [filter filter_pref]
               [lexer lexer_pref]
               [wordlist wordlist_pref]
               [storage storage_pref]
               [stoplist stoplist]
               [section group section_group]
               [memory memsize]
|   ... add attr section section_name tag tag@attr
|   add stop section tag'
```

Dynamically the clause `add attr section section_name tag tag@attr` adds an attribute section `section_name` to the existing index. You must specify the XML tag and attribute in the form `tag@attr`. You can only add attribute sections to XML section groups.

The added section, `section_name`, applies only to documents indexed after this operation. Thus for the change to take effect, you must manually re-index any

existing documents that contain the tag. The index is not rebuilt by this statement.

AUTO_ or PATH_SECTION_GROUP

When you use the `AUTO_SECTION_GROUP` or `PATH_SECTION_GROUP` to index XML documents, the system automatically creates attribute sections and names them in the form `attribute@tag`.

To search on Tale within the attribute section `booktitle`, include the following `WITHIN` clause in your `SELECT` statement:

- If you are using `XML_SECTION_GROUP`:

```
... WHERE CONTAINS ('Tale WITHIN booktitle')>0;
```
- If you are using `AUTO_ or PATH_SECTION_GROUP`

```
... WHERE CONTAINS ('Tale WITHIN title@book')>0;
```

See Also: ["Distinguishing Tags Across DocTypes"](#) on page 8-24.

Constraints for Querying Attribute Sections

The following constraints apply to querying within attribute sections:

- Regular queries on attribute text will not work unless qualified in a `WITHIN` clause. Using the following XML document:

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

querying on Tale will not work unless qualified with `'WITHIN title@book'`.
- You cannot use attribute sections in a nested `WITHIN` query.
- Phrases ignore attribute text. For example, if the original document looked like:

```
...Now is the time for all good <word type="noun"> men </word> to come to the aid.....
```

The search would result in a regular query's, "good men", and ignore the intervening attribute text.

Using INPATH or HASPATH Operators for Query Searching

Use the `INPATH` and `HASPATH` operators only when your index has been created with `PATH_SECTION_GROUP`.

Use of `PATH_SECTION_GROUP` enables path searching. Path searching extends the syntax of the `WITHIN` operator so that the section name operand (right-hand-side) is a *path* instead of a *section name*.

[Table 8–2](#) lists the different ways you can use the `INPATH` operator for path searching.

Table 8–2 Path Searching XML Documents Using the INPATH Operator

Path Search Feature	Syntax	Description
Simple Tag Searching	virginia INPATH (STATE) virginia INPATH (//STATE)	Finds all documents where the word “virginia” appears between <code><STATE></code> and <code></STATE></code> . The <code>STATE</code> element can appear at any level of the document structure.
Case-sensitivity	virginia INPATH (STATE) virginia INPATH (State)	Tags and attribute names in path searching are case-sensitive. <code>virginia INPATH STATE --</code> finds <code><STATE>virginia</STATE></code> but NOT <code><State>virginia</State></code> . To find the latter you must do <code>virginia INPATH State</code> .
Top-Level Tag Searching	virginia INPATH (Legal) virginia INPATH (/Legal)	Finds all documents where “virginia” appears in a <code>Legal</code> element which is the top-level tag. <code>Legal</code> MUST be the top-level tag of the document. <code>virginia</code> may appear anywhere in this tag - regardless of other intervening tags. For example: <pre><?xml version="1.0" standalone="yes"?> <!-- <?xmlstylesheet type="text/xsl" href="/xsl/vacourtfiling(html).xsl"? --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState>VIRGINIA</AddressState> </Address> ... </Legal></pre>

Table 8–2 Path Searching XML Documents Using the INPATH Operator

Path Search Feature	Syntax	Description
Any Level Tag Searching	virginia INPATH (//Address)	<p>'Virginia' can appear anywhere within an 'Address' tag, which may appear within any other tags. for example:</p> <pre><?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"? --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState> VIRGINIA </AddressState>... </Legal></pre>
Direct Parentage Path Searching	virginia INPATH (//CourtInformation/Location)	<p>Finds all documents where “virginia” appears in a Location element which is a direct child of a CourtInformation element. For example:</p> <pre><?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"? --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState> VIRGINIA </AddressState> </Address>... </CourtInformation></pre>

Table 8–2 Path Searching XML Documents Using the INPATH Operator

Path Search Feature	Syntax	Description
Single-Level Wildcard Searching	virginia INPATH(A/*/B) 'virginia INPATH (/ /CaseCaption/*/Location)'	<p>Finds all documents where “virginia” appears in a B element which is a grandchild of an A element. For instance, <A><D>virginia</D>. The intermediate element does not need to be an indexed XML tag. For example:</p> <pre> <?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="/xsl/vacourtfiling(html).xsl"? --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState>VIRGINIA</AddressState>... </Legal> </pre>

Table 8–2 Path Searching XML Documents Using the INPATH Operator

Path Search Feature	Syntax	Description
Multi-level Wildcard Searching	'virginia INPATH (Legal/*/Filing/*/*/CourtInformation)'	<p>'Legal' must be a top-level tag, and there must be exactly one tag-level between 'Legal' and 'Filing', and two between 'Filing' and 'CourtInformation'. 'Virginia' may then appear anywhere within 'CourtInformation'. For example:</p> <pre><?xml version="1.0" standalone="yes"?> <!-- <?xml-stylesheet type="text/xsl" href="./xsl/vacourtfiling(html).xsl"? --> <Legal> <CourtFiling> <Filing ID="f001" FilingType="Civil"> <LeadDocument> <CaseCaption> <CourtInformation> <Location> <Address> <AddressState>VIRGINIA</AddressState> </Address> </Location> <CourtName> IN THE CIRCUIT COURT OF LOUDOUN COUNTY </CourtName> </CourtInformation>....</pre>
Descendant Searching	virginia INPATH(A//B)	Finds all documents where “virginia” appears in a B element which is some descendant (any level) of an A element.
Attribute Searching	virginia INPATH(A/@B)	Finds all documents where “virginia” appears in the B attribute of an A element.
Descendant/Attribute Existence Testing	virginia INPATH (A[B])	<p>Finds all documents where “virginia” appears in an A element which has a B element as a direct child.</p> <ul style="list-style-type: none"> ■ virginia INPATH A[//B] -- Finds all documents where “virginia” appears in an A element which has a B element as a descendant (any level). ■ virginia INPATH A[@B] -- Finds all documents where “virginia” appears in an A element which has a B attribute

Table 8–2 Path Searching XML Documents Using the INPATH Operator

Path Search Feature	Syntax	Description
Attribute Value Testing	virginia INPATH A[@B = "foo"]	<p>Finds all documents where “virginia” appears in an A element which has a B attribute whose value is “foo”.</p> <ul style="list-style-type: none"> Only equality is supported as a test. Range operators and functions are not supported. The left-hand-side of the equality MUST be an attribute or tag. Literals here are not allowed. The right-hand-side must be a literal. Tags and attributes here are not allowed.
Within Equality	<p>That means that:</p> <p>virginia INPATH (A[@B = "pot of gold"])</p> <p>would, with the default lexer and stoplist, match any of the following:</p> <pre>virginia</pre> <p>By default, lexing is case-independent, so “pot” matches “POT”, virginia</p> <p>By default, “of” is a stopword, and, in a query, would match any word in that position, virginia</p>	<p>Within equality (See "Using the HASPATH Operator for Path Searching" on page 8-16) is used to evaluate the test.</p> <p>Whitespace is mainly ignored in text indexing. Again, lexing is case-independent:</p> <pre>virginia</pre> <p>Underscore is a non-alphabetic character, and is not a join character by default. As a result, it is treated more or less as whitespace and breaks up that string into three words.</p>

Table 8–2 Path Searching XML Documents Using the INPATH Operator

Path Search Feature	Syntax	Description
Numeric Equality	virginia INPATH (A[@B = 5])	Numeric literals are allowed. But they are treated as text. The within equality is used to evaluate. This means that the query does NOT match. That is, virginia does not match A[@B=5] where "5.0", a decimal is not considered the same as 5, an integer.
Conjunctive Testing	virginia INPATH (A[B AND C]) virginia INPATH (A[B AND @C = "foo"])	Predicates can be conjunctively combined.
Combining Path and Node Tests	virginia INPATH (A[@B = "foo"]/C/D) virginia INPATH(A//B[@C]/D[E])	Node tests can be applied to any node in the path.

Using the HASPATH Operator for Path Searching

Note: The HASPATH operator functions in a similar fashion to the Existsnode() operator in XMLType. See Also [Chapter 5, "Database Support for XML"](#).

Only use the HASPATH operator when your index has been created with the PATH_SECTION_GROUP. The syntax for the HASPATH operator is:

- WHERE CONTAINS(column, 'HASPATH(path)'...):** Here HASPATH searches an XML document set and returns a score of 100 for all documents where path exists. Parent and child paths are separated with the / character, for example, A/B/C. For example, the query:

```
...WHERE CONTAINS (col, 'HASPATH(A/B/C)')>0;
```

finds and returns a score of 100 for the document:

```
<A><B><C>Virginia</C></B></A>
```

without having to reference Virginia at all.

- WHERE CONTAINS(column, 'HASPATH(A="value")'...):** Here the HASPATH clause searches an XML document set and returns a score of 100 for all documents that have element A with content value and only that value.

HASPATH is used to test equality. This is the "Section Equality Testing" feature of the HASPATH operator. The query:

```
...WHERE CONTAINS virginia INPATH A
```

finds <A>virginia, but it also finds <A>virginia state. To limit the query to the term virginia and nothing else, you can use a section equality test with the HASPATH operator. For example:

```
... WHERE CONTAINS (col,'HASPATH(A="virginia")'
```

finds and returns a score of 100 only for the first document, and not the second.

Using Oracle Text to Search XML Documents

To use Oracle Text to search and retrieve data from XML documents you must do the following overall tasks:

1. Create a section group
2. Create an Oracle Text index based on the section group you created
3. Build your query application using the CONTAINS operator

Before you create a section group and Oracle text index you must first determine the role you will need and grant the appropriate privilege. See "[Oracle Text Users and Roles](#)" on page 8-5, and grant the appropriate privilege.

After creating and preparing your data, you are ready to perform the next step. See [Step 1. Create a Section Preference](#).

Using the section preference created, you then create an Oracle Text index. See [Step 2. Create an Index Using the Section Preference Created in Step 1](#).

Now you can finish building your query application. See "[Using Oracle Text to Search XML Documents](#)".

First determine the role you need. See *Oracle9i Text Reference* and "[Oracle Text Users and Roles](#)" on page 8-5, and grant the appropriate privilege as follows:

```
CONNECT system/manager
GRANT ctxapp to scott;
CONNECT scott/tiger
```

Step 1. Create a Section Preference

Here we describe the basics of how to create section preferences using `PATH_SECTION_GROUP`, `XML_SECTION_GROUP`, and `AUTO_SECTION_GROUP`.

[Table 8–3](#) describes the section groups you can use when indexing XML documents.

Table 8–3 Comparing Oracle Text Section Groups

Section Group	Description
<code>XML_SECTION_GROUP</code>	Use this group type for indexing XML documents and for defining sections in XML documents.
<code>AUTO_SECTION_GROUP</code>	<p>Use this group type to automatically create a zone section for each start-tag/end-tag pair in an XML document. The section names derived from XML tags are case-sensitive as in XML. Attribute sections are created automatically for XML tags that have attributes. Attribute sections are named in the form <code>attribute@tag</code>. Stop sections, empty tags, processing instructions, and comments are not indexed. The following limitations apply to automatic section groups:</p> <ul style="list-style-type: none">■ You cannot add zone, field or special sections to an automatic section group.■ Automatic sectioning does not index XML document types (root elements.) However, you can define stop-sections with document type.■ The length of the indexed tags including prefix and namespace cannot exceed 64 characters. Tags longer than this are not indexed.
<code>PATH_SECTION_GROUP</code>	<p>Use this group type to index XML documents. Behaves like the <code>AUTO_SECTION_GROUP</code>. With this section group you can do path searching with the <code>INPATH</code> and <code>HASPATH</code> operators. Queries are case-sensitive for tag and attribute names.</p> <p>How is <code>PATH_SECTION_GROUP</code> Similar to <code>AUTO_SECTION_GROUP</code>?</p> <p>Documents are assumed to be XML, Every tag and every attribute is indexed by default, Stop sections can be added to prevent certain tags from being indexed, Only stop sections can be added -- <code>ZONE</code>, <code>FIELD</code>, and <code>SPECIAL</code> sections cannot be added, When indexing XML document collections, you do not need to explicitly define sections as Oracle automatically does this for you.</p> <p>How Does <code>PATH_SECTION_GROUP</code> Differ From <code>AUTO_SECTION_GROUP</code>?</p> <p>Path Searching is allowed at query time (see "Case Study: Searching an Online FAQ List Using Oracle Text" and "Using the HASPATH Operator for Path Searching" on page 8-16) with the new <code>INPATH</code> and <code>HASPATH</code> operators, Tag and attribute names are case-sensitive in queries.</p>

Note: If you are using the `AUTO_SECTION_GROUP` or `PATH_SECTION_GROUP` to index an XML document collection, you need not explicitly define sections since the system does this for you during indexing.

Deciding Which Section Group to Use

How do you determine which section groups is best for your application? This depends on your application. [Table 8–4](#) lists some general guidelines to help you decide which of the `XML_`, `AUTO_`, or `PATH_` section groups to use when indexing your XML documents, and why.

Table 8–4 Guidelines for Choosing `XML_`, `AUTO_`, or `PATH_` Section Groups

Application Criteria	<code>XML_section_...</code>	<code>AUTO_section_...</code>	<code>PATH_section_...</code>
You are using XPATH search features			X
You know the layout and structure of your XML documents, and you can predefine the sections on which users are most likely to search.	X		
You do not know which tags users are most likely to search.		X	
Query performance, in general	Fastest	Little slower than <code>XML_section_...</code>	Little slower than <code>AUTO_section_...</code>
Indexing performance, in general	Fastest	Little slower than <code>XML_section_...</code>	Little slower than <code>AUTO_section_...</code>
Index size	Smallest	Little larger than <code>XML_section_...</code>	Little larger than <code>AUTO_section_...</code>
Other features	Mappings can be defined so that tags in one or different DTDs can be mapped to one section. Good for DTD evolution and data aggregation.	Simplest. No need to define mapping, <code>add_stop_section</code> can be used to ignore some sections.	Designed for more sophisticated XPATH- like queries

Creating a Section Preference with XML_SECTION_GROUP

The following command creates a section group called, `xmlgroup`, with the `XML_SECTION_GROUP` group type.

```
EXEC ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
```

You can add sections to this group using `CTX_DDL.ADD_SECTION`. Consider an XML file that defines the `BOOK` tag with a `TITLE` attribute as follows:

```
<BOOK TITLE="Tale of Two Cities"> It was the best of times. </BOOK>
```

To define the title attribute as an attribute section, create an `XML_SECTION_GROUP` and define the attribute section as follows:

```
EXEC ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');
```

When you define the `TITLE` attribute section as such and index the document set, you can query the XML attribute text as follows:

```
'Cities within booktitle'
```

Creating a Section Preference with AUTO_SECTION_GROUP

You can set up your indexing operation to automatically create sections from XML documents using the section group `AUTO_SECTION_GROUP`. Here, Oracle creates zone sections for XML tags. Attribute sections are created for those tags that have attributes, and these attribute sections are named in the form “tag@attribute.”

The following command creates a section group called `autogroup` with the `AUTO_SECTION_GROUP` group type. This section group *automatically* creates sections from tags in XML documents.

```
EXEC ctx_ddl.create_section_group('autogroup', 'AUTO_SECTION_GROUP');
```

Note: You can add attribute sections only to XML section groups. When you use `AUTO_SECTION_GROUP`, attribute sections are created automatically. Attribute sections created automatically are named in the form `tag@attribute`.

Creating a Section Preference with PATH_SECTION_GROUP

To enable path section searching, index your XML document with `PATH_SECTION_GROUP`. For example:

```
EXEC ctx_ddl.create_section_group('xmlpathgroup', 'PATH_SECTION_GROUP');
```

Step 2. Create an Index Using the Section Preference Created in Step 1

Create an index depending on which section group you used to create a preference:

Creating an Index Using XML_SECTION_GROUP

To index your XML document when you have used XML_SECTION_GROUP, you can use the following statement:

```
CREATE INDEX myindex ON docs(htmlfile) INDEXTYPE IS ctxsys.context
  parameters('section group xmlgroup');
```

See Also: ["Oracle Text Example 1: Creating an Index Using XML_SECTION_GROUP"](#) on page 8-21.

Creating an Index Using AUTO_SECTION_GROUP

The following statement creates the index, myindex, on a column containing XML files using the AUTO_SECTION_GROUP:

```
CREATE INDEX myindex ON xmldocs(xmlfile) INDEXTYPE IS ctxsys.context PARAMETERS
('section group autogroup');
```

Creating an Index Using PATH_SECTION_GROUP

To index your XML document when you have used PATH_SECTION_GROUP, you can use the following statement:

```
CREATE INDEX myindex ON xmldocs(xmlfile) INDEXTYPE IS ctxsys.context PARAMETERS
('section group xmlpathgroup');
```

See Also: *Oracle9i Text Reference* for detailed notes on CTX_DDL.

Oracle Text Example 1: Creating an Index Using XML_SECTION_GROUP

```
EXEC ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');

/* ADDING A FIELD SECTION */
EXEC ctx_ddl.Add_Field_Section /* THIS IS KEY */
  ( group_name   =>'my_section_group',
    section_name =>'author', /* do this for EVERY tag used after "WITHIN" */
    tag          =>'author'
```

```
);

EXEC ctx_ddl.Add_Field_Section /* THIS IS KEY */
  ( group_name   =>'my_section_group',
    section_name =>'document', /*do this for EVERY tag after "WITHIN" */
    tag          =>'document'
  );

...
/
/* ADDING AN ATTRIBUTE SECTION */
EXEC ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');

/* The more sections you add to your index, the longer your search will take.*/
/* Useful for defining attributes in XML documents as sections. This allows*/
/* you to search XML attribute text using the WITHIN operator.*/
/* The section name:
/* ** Is used for WITHIN queries on the attribute text.
/* ** Cannot contain the colon (:) or dot (.) characters.
/* ** Must be unique within group_name.
/* ** Is case-insensitive.
/* ** Can be no more than 64 bytes.
/* ** The tag specifies the name of the attribute in tag@attr format. This is
/* case-sensitive. */
/* Names used as arguments of the keyword WITHIN can be different from the
/* actual XML tag names. Many tags can be mapped to the same name at query
/* time.*/

/* ADDING A ZONE SECTION */
/* If You have an XML document that contains the <book> tag declared for */
/* different document types. You can create a distinct book section for each */
/* document type. If mydocname is declared in your DTD as an XML document */
/* type (root element) as follows: */

<!DOCTYPE mydocname ... [...

/* Within mydocname, element <book> is declared. For this tag, you can create */
/* a section named, mybooksec, that's sensitive to the tag's document type as */
/* follows: */

EXEC ctx_ddl.add_zone_section('myxmlgroup', 'mybooksec', 'mydocname(book)');

/* Call CTX_DDL.Add_Zone_Section for each tag in your XML document that you need
/* to search on. */
```

```

CREATE INDEX my_index ON my_table ( my_column )
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'SECTION GROUP my_section_group' );

SELECT my_column FROM my_table
  WHERE CONTAINS(my_column, 'smith WITHIN author') > 0;

```

Oracle Text Example 2: Creating an Index Using AUTO_SECTION_GROUP

```

ctx_ddl_create_section_group('auto', 'AUTO_SECTION_GROUP');

CREATE INDEX myindex ON docs(xmlfile_column)
  INDEXTYPE IS ctxsys.context
  PARAMETERS ('filter ctxsys.null_filter SECTION GROUP auto');

SELECT xmlfile_column FROM docs
  WHERE CONTAINS (xmlfile_column, 'virginia WITHIN title')>0;

```

Oracle Text Example 3: Creating an Index Using PATH_SECTION_GROUP

```

EXEC ctx_ddl.create_section_group('xmlpathgroup', 'PATH_SECTION_GROUP');

CREATE INDEX myindex ON xmldocs(xmlfile_column)
  INDEXTYPE IS ctxsys.context
  PARAMETERS ('section group xmlpathgroup');

SELECT xmlfile_column FROM xmldocs
... WHERE CONTAINS (column, 'Tale WITHIN title@book')>0;

```

Building XML Query Applications with Oracle Text

Building XML query applications with Oracle Text includes the following topics:

- [Querying Within Attribute Sections](#)
- [Querying XML Documents](#)
- [Procedure for Building a Query Application with Oracle Text](#)
- [Creating Sections in XML Documents that are Document Type Sensitive](#)

See Also:

- *Oracle9i Text Developer's Guide*
- *Oracle9i Text Reference*
- ["Case Study: Searching an Online FAQ List Using Oracle Text"](#) on page 8-41.

Querying XML Documents

Distinguishing Tags Across DocTypes

In previous releases, the XML section group was unable to distinguish between tags in different DTD's. For instance, perhaps you have a DTD for storing contact information:

```
<!DOCTYPE contact>
<contact>
  <address>506 Blue Pool Road</address>
  <email>dudeman@radical.com</email>
</contact>
```

Appropriate sections might look like:

```
ctx_ddl.add_field_section('mysg','email','email');
ctx_ddl.add_field_section('mysg','address','address');
```

This is fine until you have a different kind of document in the same table:

```
<!DOCTYPE mail>
<mail>
  <address>dudeman@radical.com</address>
</mail>
```

Now your address section, originally intended for street addresses, starts picking up email addresses, because of tag collision.

Specifying Doctype Limiters to Distinguish Between Tags

Oracle8i release 8.1.5 and higher allow you to *specify doctype limiters* to distinguish between these tags across doctypes. Simply specify the doctype in parentheses before the tag as follows:

```
ctx_ddl.add_field_section('mysg','email','email');
```

```
ctx_ddl.add_field_section('mysg','address','(contact)address');
ctx_ddl.add_field_section('mysg','email','(mail)address');
```

Now when the XML section group sees an address tag, it will index it as the address section when the document type is `contact`, or as the email section when the document type is `mail`.

Doctype-Limited and Unlimited Tags in a Section Group

If you have both doctype-limited and unlimited tags in a section group:

```
ctx_ddl.add_field_section('mysg','sec1','(type1)tag1');
ctx_ddl.add_field_section('mysg','sec2','tag1');
```

Then the limited tag applies when in the doctype, and the unlimited tag applies in all other doctypes.

Querying is unaffected by this -- the query is done on the section name, not the tag, so querying for an email address would be done like:

```
radical WITHIN email
```

which, since we have mapped two different kinds of tags to the same section name, finds documents independent of which tags are used to express the email address.

Querying Within Attribute Sections

You can query within attribute sections when you *index* with either of the following as your section group type:

- XML_SECTION_GROUP
- AUTOMATIC_SECTION_GROUP
- PATH_SECTION_GROUP

Consider the following XML document:

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

You can define the section `title@book` to be the attribute section `title`. You can do so with the `CTX_DLL.ADD_ATTR_SECTION` procedure or dynamically after indexing with `ALTER INDEX`.

Note:

- When you use the `AUTO_SECTION_GROUP` to index XML documents, the system automatically creates attribute sections and names them in the form `attribute@tag`.
 - If you use the `XML_SECTION_GROUP`, you can name attribute sections anything with `CTX_DDL.ADD_ATTR_SECTION`.
-
-

To search on Tale within the attribute section title, you issue the following query:

```
'Tale WITHIN BOOK@TITLE'
```

You cannot use attribute sections in a nested `WITHIN` query.

Phrases ignore attribute text. For example, if the original document looked like:

Now is the time for all good `<word type="noun"> men </word>` to come to the aid. The `WITHIN` operator requires you to know the name of the section you search. A list of defined sections can be obtained using the `CTX_SECTIONS` or `CTX_USER_SECTIONS` views.

See Also:

- Oracle9i Text Developer's Guide
- *Oracle9i Text Reference*

XML_SECTION_GROUP Attribute Sections

In Oracle8i Release 1(8.1.5) and higher, `XML_SECTION_GROUP` offers the ability to index and search within *attribute* values. Consider a document with the following lines:

```
<comment author="jeeves">
  I really like Oracle Text
</comment>
```

Now `XML_SECTION_GROUP` offers an attribute section. This allows the inclusion of attribute values to index. For example:

```
ctx_ddl.add_attr_section('mysg','author','comment@author');
```


The syntax is similar to other `add_section` calls. The first argument is the name of the section group, the second is the name of the section, and the third is the tag, in the form `<tag_name>@<attribute_name>`. This tells Oracle Text to index the contents of the `author` attribute of the `comment` tag as the section “author”.

Query syntax is just like for any other section:

```
WHERE CONTAINS ( ... , 'jeeves WITHIN author...', ...)...
```

and finds the document.

Attribute Value Sensitive Section Search

Attribute sections allow you to search the contents of attributes. They do not allow you to use attribute values to specify sections to search. For instance, given the document:

```
<comment author="jeeves">
  I really like Oracle Text
</comment>
```

You can find this document by asking:

```
jeeves within comment@author
```

which is equivalent to “find me all documents which have a `comment` element whose `author` attribute’s value includes the word `jeeves`”.

However, there you cannot currently request the following:

```
interMedia within comment where (@author = "jeeves")
```

in other words, “find me all documents where `interMedia` appears in a `comment` element whose `author` is `jeeves`”. This feature -- attribute value sensitive section searching -- is planned for future versions of the product.

Dynamic Add Section

Because the section group is defined before creating the index, Oracle8i Release 1 (8.1.5) is limited in its ability to cope with changing structured document sets; if your documents start coming with new tags, or you start getting new doctypes, you have to re-create the index to start making use of those tags.

In Oracle8i Release 2 (8.1.6) and higher allows you to add new sections to an existing index without rebuilding the index, using `alter index` and the new `add section` parameters string syntax:

```
add zone section <section_name> tag <tag>
add field section <section_name> tag <tag> [ visible | invisible ]
```

For instance, to add a new zone section named tsec using the tag title:

```
alter index <indexname> rebuild
parameters ('add zone section tsec tag title')
```

To add a new field section named asec using the tag author:

```
alter index <indexname> rebuild
parameters ('add field section asec tag author')
```

This field section would be invisible by default, just like when using `add_field_section`. To add it as visible field section:

```
alter index <indexname> rebuild
parameters ('add field section asec tag author visible')
```

Dynamic add section only modifies the index meta-data, and does not rebuild the index in any way. This means that these sections take effect for any document indexed after the operation, and do not affect any existing documents -- if the index already has documents with these sections, they must be manually marked for re-indexing (usually with an update of the indexed column to itself).

This operation does not support addition of special sections. Those would require all documents to be re-indexed, anyway. This operation cannot be done using `rebuild online`, but it should be a fairly quick operation.

Constraints for Querying Attribute Sections

The following constraints apply to querying within attribute sections:

- Regular queries on attribute text do not hit the document unless qualified in a `within` clause. Assume you have an XML document as follows:

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

A query on `Tale` by itself does not produce a hit on the document unless qualified with `WITHIN title@book`. This behavior is like field sections when you set the `visible` flag set to `false`.

- You cannot use attribute sections in a nested `WITHIN` query.
- Phrases ignore attribute text. For example, if the original document looked like:

```
Now is the time for all good <word type="noun"> men </word> to come to the
```

aid.

Then this document would hit on the regular query good men, ignoring the intervening attribute text.

WITHIN queries can distinguish repeated attribute sections. This behavior is like zone sections but unlike field sections. For example, for the following document:

```
<book title="Tale of Two Cities">It was the best of times.</book>
<book title="Of Human Bondage">The sky broke dull and gray.</book>
```

Assume the book is a zone section and book@author is an attribute section. Consider the query:

```
'(Tale and Bondage) WITHIN book@author'
```

This query does not hit the document, because tale and bondage are in different occurrences of the attribute section book@author.

Procedure for Building a Query Application with Oracle Text

To build the query application with Oracle Text carry out the indexing steps first..

The next step is to build your query application. To do so follow these steps:

1. Create a preference using the procedure, CTX_DDL.create_preference. See "[Step 1. Create a Preference](#)"
2. Set preference's attributes using CTX_DDL.Add_Attr_Section and so on. See "[Step 2. Set the Preference's Attributes](#)".
3. Create your query syntax

You can query within attribute sections when you index with either XML_SECTION_GROUP or AUTOMATIC_SECTION_GROUP as your section group type.

Note:

- Not everything in your document may be searchable. You must first state what is searchable using the.....add_....._section
 - The more sections you add to your index the longer the search will take!
-
-

Nested tag searching is supported in Oracle Text.

Using Table CTX_OBJECTS and CTX_OBJECT_ATTRIBUTES View

The CTX_OBJECT_ATTRIBUTES view displays attributes that can be assigned to preferences of each object. It can be queried by all users.

Check out the structure of CTX_OBJECTS and CTX_OBJECT_ATTRIBUTE view, with the following DESCRIBE commands. Because we are only interested in querying XML documents in this chapter, we focus on XML_SECTION_GROUP and AUTO_SECTION_GROUP.

Describe ctx_objects

```
SELECT obj_class, obj_name FROM ctx_objects
ORDER BY obj_class, obj_name;
```

The result is:

```
...
SECTION_GROUP          AUTO_SECTION_GROUP    <<==
SECTION_GROUP          BASIC_SECTION_GROUP
SECTION_GROUP          HTML_SECTION_GROUP
SECTION_GROUP          NEWS_SECTION_GROUP
SECTION_GROUP          NULL_SECTION_GROUP
SECTION_GROUP          XML_SECTION_GROUP    <<==
...

```

Describe ctx_object_attributes

```
SELECT oat_attribute FROM ctx_object_attributes
WHERE oat_object = 'XML_SECTION_GROUP';
```

The result is:

```
OAT_ATTRIBUTE
-----
ATTR
FIELD
SPECIAL
ZONE
```

```
SELECT oat_attribute FROM ctx_object_attributes
WHERE oat_object = 'AUTO_SECTION_GROUP';
```

The result is:

```
OAT_ATTRIBUTE
-----
STOP
```

Step 1. Create a Preference

The first thing you must do is create a preference. To do this, use the `CTX_DDL.Create_Preference` procedure. For example:

CTX_DDL.Create_Preference

```
CTX_DDL.Create_Preference (
  preference_name => 'books' /* or whatever you want to call it */
  object_name     => 'XML_SECTION_GROUP' /* either XML_SECTION_GROUP or AUTO_
SECTION_GROUP */);
```

To drop this preference use the following syntax:

```
CTX_DDL.Drop_Preference (
  preference_name => 'books');
```

Step 2. Set the Preference's Attributes

To set the preference's attributes for `XML_SECTION_GROUP`, use the following procedures:

- `Add_Zone_Section`
- `Add_Attr_Section`
- `Add_Field_Section`
- `Add_Special_Section`

To set the preference's attributes for `AUTO_SECTION_GROUP`, use the following procedures:

- `Add_Zone_Section`
- `Add_Attr_Section`
- `Add_Field_Section`

There are corresponding `CTX_DDL.drop` sections and `CTX_DDL.remove` section syntax.

2.1 Using `CTX_DDL.add_zone_section`

The syntax for `CTX_DDL.add_zone_section` follows:

```
CTX_DDL.Add_Zone_Section (
```

```
group_name      => 'my_section_group' /* whatever you called it above */
section_name    => 'author' /* what you want to call this section */
tag             => 'my_tag' /* what represents it in XML */ );
```

where 'my_tag' implies opening with <my_tag> and closing with </my_tag>.

add_zone_section Guidelines

add_zone_section guidelines are listed here:

- Call CTX_DDL.Add_Zone_Section for each tag in your XML document that you need to search on.

2.2 Using CTX_DDL.Add_Attr_Section

The syntax for CTX_DDL.add_attr_section follows:

```
CTX_DDL.Add_Attr_Section ( /* call this as many times as you need to describe
                           the attribute sections */
group_name      => 'my_section_group' /* whatever you did call it above */
section_name    => 'author' /* what you want to call this section */
tag            => 'my_tag' /* what represents it in XML */ );
```

where 'my_tag' implies opening with <my_tag> and closing with </my_tag>.

add_attr_section Guidelines

add_attr_section guidelines are listed here:

- Consider meta_data attribute author:

```
<meta_data author = "John Smith" title="How to get to Mars">
```

The more sections you add to your index, the longer your search will take.

add_attr_section adds an attribute section to an XML section group. This procedure is useful for defining attributes in XML documents as sections. This allows you to search XML *attribute* text with the WITHIN operator.

The section_name:

- Is the name used for WITHIN queries on the attribute text.
- Cannot contain the colon (:) or dot (.) characters.
- Must be unique within group_name.
- Is case-insensitive.

- Can be no more than 64 bytes.

The tag specifies the name of the attribute in tag@attr format. This is case-sensitive.

Note: In the add_attr_section procedure, you can have many tags all represented by the same section name at query time. Explained in another way, the names used as the arguments of the keyword WITHIN can be different from the actual XML tag names. That is many tags can be mapped to the same name at query time. This feature enhances query usability.

2.3 Using CTX_DDL.add_field_section

The syntax for CTX_DDL.add_field_section follows:

```
CTX_DDL.Add_Field_Section (
  group_name      => 'my_section_group' /* whatever you called it above */
  section_name    => 'qq' /* what you want to call this section */
  tag             => 'my_tag' /* what represents it in XML */ );
visible          => TRUE or FALSE );
```

add_field_section Guidelines

add_field_section guidelines are listed here:

- add_field_section and add_zone_section attributes differ in performance.
- In a document, tags can be repeated two or more times, however some tags, such as “title”, occur only once. A DTD (or XML Schema) define how many times the tags occur.
- **Visible attribute:** This is available in add_field_section but not available in the add_zone_section. If VISIBLE is set to TRUE then a query such as “... CONTAINS virginia... becomes irrelevant if cat occurs in title or paragraph.

Consider again the query, “... CONTAINS virginia...”. You may not get a hit if you use VISIBLE=TRUE. If VISIBLE=FALSE, the index will be smaller. You may lose some functionality but your performance will be improved, compared to if you set VISIBLE =TRUE.

- If you set up your index using the add_zone_section...
- If you set up your index using the add_field_section...

Note: Constructing an index is harder if you have to cater for the fact that there could be more than one occurrence of any one tag.

- If the tag in your XML document occurs only once, use the single `add_field_section` procedure. For example, "... CONTAINS virginia and state WITHIN title....."
 - If the tag in your XML document occurs more than once, use the `add_zone_section` procedure. For example, "... CONTAINS virginia and state WITHIN paragraph....". This has many possibilities.
-
-

How Attr_Section Differs From Field_Section

Attribute section differs from field section in the following ways:

- **Attribute text is considered invisible**, though, so the following:

```
WHERE CONTAINS (... , '... jeeves',... )...
```

does NOT find the document, somewhat like field sections. Unlike field sections, however, attribute section within searches can distinguish between occurrences. Consider the document:

```
<comment author="jeeves">
  I really like Oracle Text
</comment>
<comment author="bertram">
  Me too
</comment>
```

the query:

```
WHERE CONTAINS (... , '(cryil and bertram) WITHIN author', ... )...
```

will NOT find the document, because "jeeves" and "bertram" do not occur within the SAME attribute text.

- **Attribute section names cannot overlap with zone or field section names**, although you can map more than one `tag@attr` to a single section name. Attribute sections do not support default values. Given the document:

```
<!DOCTYPE foo [
  <!ELEMENT foo (bar)>
  <!ELEMENT bar (#PCDATA)>
<!ATTLIST bar
```



```

      rev CDATA "8i">
    ]>
    <foo>
      <bar>whatever</bar>
    </foo>

```

and attribute section:

```
ctx_ddl.add_attr_section('mysg', 'barrev', 'bar@rev');
```

the query:

8i within barrev does not hit the document, although in XML semantics, the “bar” element has a default value for its “rev” attribute.

2.5 Using CtX_DDL.Add_Stop_Section

```

CtX_DDL.Add_Stop_Section (
  group_name      => 'my_section_group' /* whatever you called it above */
  section_name    => 'qq' /* what you want to call this section */ );

```

Step 3. Create Your Query Syntax

See the section, ["Querying with the CONTAINS Operator"](#) for information about how to use the CONTAINS operator in query statements.

Querying Within Attribute Sections

You can query within attribute sections when you index with either XML_SECTION_GROUP or AUTO_SECTION_GROUP as your section group type.

Assume you have an XML document as follows:

```
<book title="Tale of Two Cities">It was the best of times.</book>
```

You can define the section title@book as the attribute section title. You can do so with the CTX_DLL.Add_Attr_Section procedure or dynamically after indexing with ALTER INDEX.

Note: When you use the AUTO_SECTION_GROUP to index XML documents, the system automatically creates attribute sections and names them in the form attribute@tag.

If you use the `XML_SECTION_GROUP`, you can name attribute sections anything with `CTX_DDL.Add_Attr_Section`.

To search on Tale within the attribute section title, issue the following query:

```
WHERE CONTAINS (...,'Tale WITHIN title', ...)
```

Using `XML_SECTION_GROUP` and `add_attr_section` to Aid Querying

Consider an XML file that defines the `BOOK` tag with a `TITLE` attribute as follows:

```
<BOOK TITLE="Tale of Two Cities">
It was the best of times. </BOOK>
<Author="Charles Dickens">
Born in England in the town, Stratford_Upon_Avon </Author>
```

Recall the `CTX_DDL.Add_Attr_Section` syntax is:

```
CTX_DDL.Add_Attr_Section ( group_name, section_name, tag );
```

To define the title attribute as an attribute section, create an `XML_SECTION_GROUP` and define the attribute section as follows:

```
ctx_ddl_create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_attr_section('myxmlgroup', 'booktitle', 'book@title');
ctx_ddl.add_attr_section('myxmlgroup', 'authors', 'author');
end;
```

When you define the `TITLE` attribute section as such and index the document set, you can query the XML attribute text as follows:

```
... WHERE CONTAINS (...,'Cities WITHIN booktitle', ....)...
```

When you define the `AUTHOR` attribute section as such and index the document set, you can query the XML attribute text as follows:

```
... WHERE 'England WITHIN authors'
```

Oracle Text Example 4: Querying a... Document

This example does the following:

1. Creates and populates table `res_xml`
2. Creates an index, `section_group`, and preferences
3. Paramaterizes the preferences
4. Runs a test query against `res_xml`

```

drop table res_xml;

CREATE TABLE res_xml (
  pk          NUMBER PRIMARY KEY ,
  text       CLOB
) ;

insert into res_xml values(111,
  'ENTITY chap8 "Chapter 8, <q>Keeping it Tidy: the XML Rule Book </q>" this is
the document section!');
commit;

---
--- script to create index on res_xml
---

--- cleanup, in case we have run this before
DROP INDEX res_index ;
EXEC CTX_DDL.DROP_SECTION_GROUP ( 'res_sections' ) ;

--- create a section group
BEGIN
  CTX_DDL.CREATE_SECTION_GROUP ( 'res_sections', 'XML_SECTION_GROUP' ) ;
  CTX_DDL.ADD_FIELD_SECTION ( 'res_sections', 'chap8', '<q>' ) ;
END ;
/

begin
  ctx_ddl.create_preference
  (
    preference_name => 'my_basic_lexer',
    object_name     => 'basic_lexer'
  );
  ctx_ddl.set_attribute
  (
    preference_name => 'my_basic_lexer',
    attribute_name  => 'index_text',
    attribute_value => 'true'
  );
  ctx_ddl.set_attribute
  (
    preference_name => 'my_basic_lexer',
    attribute_name  => 'index_themes',
    attribute_value => 'false');
end;

```

```
/

CREATE INDEX res_index
  ON res_xml(text)
  INDEXTYPE IS ctxsys.context
  PARAMETERS ( 'lexer my_basic_lexer SECTION GROUP res_sections' ) ;
```

Test the above index with a test query, such as:

```
SELECT pk FROM res_xml WHERE CONTAINS( text, 'keeping WITHIN chap8' )>0 ;
```

Oracle Text Example 5: Creating an Index and Performing a Text Query

Creating Table explain_ex to Use in this Example

```
drop table explain_ex;

create table explain_ex
(
  id          number primary key,
  text       varchar(2000)
);

insert into explain_ex ( id, text )
  values ( 1, 'thinks thinking thought go going goes gone went' || chr(10) ||
          'oracle orackle oricle dog cat bird' || chr(10) ||
          'President Clinton' );
insert into explain_ex ( id, text )
  values ( 2, 'Last summer I went to New England' || chr(10) ||
          'I hiked a lot.' || chr(10) ||
          'I camped a bit.' );

commit;
```

Text Query Using "ABOUT" in the Text Query Expression

```
Set Define Off
select text
  from explain_ex
 WHERE CONTAINS ( text,
  '( $( think & go ) , ?oracle ) & ( dog , ( cat & bird ) ) & about(mammal
                                     during Bill Clinton)' ) > 0;

select text
  from explain_ex
```

```
WHERE CONTAINS ( text, 'about ( camping and hiking in new england )' ) > 0;
```

Creating Sections in XML Documents that are Document Type Sensitive

Consider an XML document set that contains the <book> tag declared for different document types. You need to create a distinct book section for each document type. Assume that mydocname is declared as an XML document type (root element) as follows:

```
<!DOCTYPE mydocname ... [...
```

Within mydocname, the element <book> is declared. For this tag, you can create a section named mybooksec that is sensitive to the tag's document type as follows:

```
begin
ctx_ddl.create_section_group('myxmlgroup', 'XML_SECTION_GROUP');
ctx_ddl.add_zone_section('myxmlgroup', 'mybooksec', 'mydocname(book)');
end;
```

Note:

- Oracle knows what the end tags look like from the group_type parameter you specify when you create the section group. The start tag you specify must be unique within a section group.
 - Section names need not be unique across tags. You can assign the same section name to more than one tag, making details transparent to searches.
-
-

Repeated Sections

Zone sections can repeat. Each occurrence is treated as a separate section. For example, if <H1> denotes a heading section, they can repeat in the same documents as follows:

```
<H1> The Brown Fox </H1>
<H1> The Gray Wolf </H1>
```

Assuming that these zone sections are named Heading.

The query:

```
WHERE CONTAINS (... , 'Brown WITHIN Heading', ...)...
```

returns this document.

But the query:

```
WHERE CONTAINS (...,' (Brown and Gray) WITHIN Heading',...)
```

does not.

Overlapping Sections

Zone sections can overlap each other. For example, if `` and `<I>` denote two different zone sections, they can overlap in document as follows:

```
plain <B> bold <I> bold and italic </B> only italic </I> plain
```

Nested Sections

Zone sections can nest, including themselves as follows:

```
<TD>
  <TABLE>
    <TD>nested cell</TD>
  </TABLE>
</TD>
```

Using the `WITHIN` operator, you can write queries to search for text in sections within sections.

Nested Section Query Example

For example, assume the `BOOK1`, `BOOK2`, and `AUTHOR` zone sections occur as follows in documents `doc1` and `doc2`:

`doc1`:

```
<book1><author>Scott Tiger</author> This is a cool book to read.</book1>
```

`doc2`:

```
<book2> <author>Scott Tiger</author> This is a great book to read.</book2>
```

Consider the nested query:

```
'Scott WITHIN author WITHIN book1'
```

This query returns only `doc1`.

Presenting the Results of Your Query

A Text query application allows you to view the documents returned by a query. You typically select a document from the hitlist and then your application presents the document in some form.

With Oracle Text, you can render a document in different ways. For example, with the query terms highlighted. Highlighted query terms can be either the words of a word query or the themes of an ABOUT query in English. This rendering uses the CTX_DOC.HIGHLIGHT or CTX_DOC.MARKUP procedures.

You can also obtain theme information from documents with the CTX_DOC.THEMES PL/SQL package. Besides these there are several other CTX_DOC procedures for presenting your query results.

See Also: *Oracle9i Text Reference* for more information on the CTX_DOC package.

Case Study: Searching an Online FAQ List Using Oracle Text

Note: ■ You can download this sample application from <http://otn.oracle.com/products/text>.

Consider the scenario where your company has several FAQs for each product. Each FAQ is an XML document similar to the following:

```
<?xml version="1.0"?>
  <FAQ OWNER="Billy Text">
    <TITLE>Oracle Text FAQ</TITLE>
    <DESCRIPTION>Everything you always wanted to know ...</DESCRIPTION>
    <QUESTION>What is Oracle Text?</QUESTION>
    <ANSWER>Oracle Text uses standard SQL to index, ...</ANSWER>
  </FAQ>
```

In this sample FAQ case study, we are only using 3 FAQs, with titles: Text, Performance, and XML. The sample FAQ program searches for information within the FAQs' XML tags using the WITHIN operator. The pull-down menu for the FAQ user interface is generated at run-time. [Figure 8-1](#) shows the online FAQ Search user interface. The pull down menu shows the XML elements selected for use in searching the FAQ data:

- Title

- Question
- FAQ
- Description
- Answer

"FAQ Owner" is actually an element *attribute*. Attributes are also searchable. These tags or elements are used here to assist users in fine tuning their keyword search for desired FAQs.

Figure 8–1 Online FAQ Search User Interface: Search Options



Figure 8–2 shows to enter a search for keyword “XML”, in all TITLE elements of the FAQs. The result of the search here is one FAQ with title, “XML...”

Figure 8–2 Creating an Online FAQ Search User Interface with Oracle Text: Searching for “XML” Within TITLE

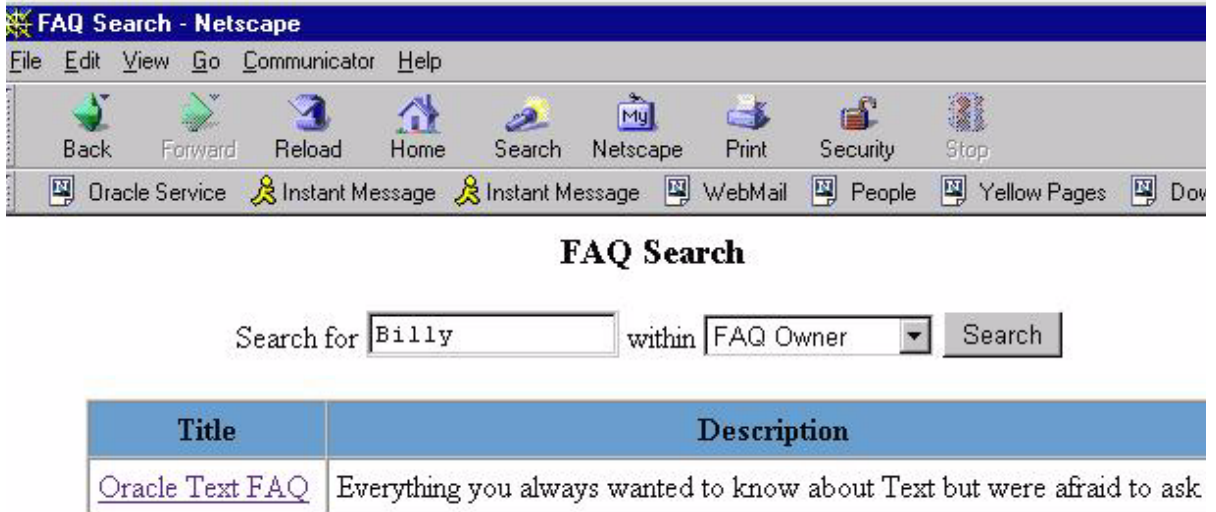


Figure 8–3 shows how you can enter an attribute search for “Billy” within FAQ OWNER by using TITLE, where TITLE is the *attribute* of the element FAQ OWNER. The syntax is “...WHERE Billy WITHIN faq@owner”.

Figure 8–3 *Creating an Online FAQ Search User Interface with Oracle Text: Attribute Searching for “Billy” Within “FAQ OWNER”*



To create your online FAQ search program for your company, follow these steps:

[1 Create and Populate Your FAQ Table. Create an Auto Section Group and Text Index](#)

[2 Compile showxml.psp](#)

[3 Compile faqsearch.psp](#)

1 Create and Populate Your FAQ Table. Create an Auto Section Group and Text Index

Run `faqsearch_install.sql`. This script does the following:

- Creates the table `faq`.
- Populates table `faq` with three records of data.
- Creates the section group and the Text index.

`faqsearch_install.sql`

```
insert into faq(tk,xml_desc)
  values(1,'<?xml version="1.0"?>
<FAQ OWNER="Billy Text">
<TITLE>Oracle Text FAQ</TITLE><DESCRIPTION>Everything you always wanted to know
about Text but were afraid to ask</DESCRIPTION>
<QUESTION>What is Oracle Text?</QUESTION>
```

```
<ANSWER>Oracle Text uses standard SQL to index, search, and analyze text and
documents stored in the database, files or websites</ANSWER>
```

```
<QUESTION>What is ABOUT?</QUESTION>
```

```
<ANSWER>ABOUT queries increase the number of relevant documents returned by
a query.</ANSWER>');
```

```
insert into faq(tk,xml_desc)
```

```
  values(2,'<?xml version="1.0"?><FAQ OWNER="Jack Performance">
```

```
<TITLE>Text Performance Guide</TITLE>
```

```
<DESCRIPTION>Oracle Text and interMedia Text performance guide</DESCRIPTION>
```

```
<QUESTION>What do we mean by query performance anyway?</QUESTION>
```

```
<ANSWER>There are generally two measures of query performance - response
time (the time to get an answer to an individual query), and throughput (the
number of queries that can be run in any time period, eg queries per
second).</ANSWER></FAQ>');
```

```
insert into faq(tk,xml_desc)
```

```
  values(3,'<?xml version="1.0"?><FAQ OWNER="John XML">
```

```
<TITLE>XML FAQ</TITLE><DESCRIPTION>Oracle XML FAQ</DESCRIPTION>
```

```
<QUESTION>What is XML?</QUESTION>
```

```
<ANSWER>XML stands for eXtensible Markup Language. XML s quickly becoming
the standard way to identify and describe data on the web because it has proved
broadly implementable and easy to deploy</ANSWER>
```

```
<QUESTION>What is the Oracle Kit?</QUESTION>
```

```
<ANSWER>The Oracle XML Developer Kit (XDK) contains the basic building
blocks for reading, manipulating, transforming and viewing XML documents. To
provide a broad variety of deployment options, the Oracle XDK is available for
Java, C, C++ and PL/SQL.</ANSWER></FAQ>');
```

```
commit;
```

```
begin
```

```
  ctx_ddl.create_section_group('faq_auto_section_group','auto_section_group');
```

```
end;
```

```
/
```

```
create index faq_idx on faq(xml_desc) indextype is ctxsys.context
```

```
  parameters('section group faq_auto_section_group')
```

```
/
```

2 Compile showxml.psp

To develop and deploy PL/SQL Server Pages (PSP), you need the Oracle server at version 8.1.6 or later, together with a PL/SQL web gateway. Currently, the Web gateways are Oracle Application Server, the WebDB PL/SQL Gateway, and the OAS PL/SQL Cartridge. Before you start with PSP, you should have access to both the database server and the web server for one of these gateways.

To compile `showxml.psp`, use the command:

```
loadpsp -replace -user username/passwd showxml.psp
```

See Also: *Oracle9i Application Developer's Guide - Fundamentals* for information about using PSP programs.

showxml.psp

```
<%@ plsql procedure="showxml" %>
<%@ plsql parameter="id" default="null" %>
<%! v_text          varchar2(32767); %>
<%! v_text_xml      varchar2(32767); %>

<%
    select xml_desc into v_text from faq where tk=id;
%>

<html>
<title>Show xml </title>
<body>
<h3>XML Content</h3>
<hr>

    <pre>
    <% v_text_xml := replace(v_text,'<','&lt;'); %>

    <%= v_text_xml %>
    </pre>

</body>
</html>
```

3 Compile faqsearch.psp

Open a URL in your browser to access `faqsearch.psp`, as follows:

`http://myserver_and_directory/faqsearch`

faqsearch.psp

```
<%@ plsql parameter="query" default="null" %>
<%@ plsql parameter="tagvalue" default="null" %>

<%! v_results numeric := 0; %>

<html>
<head>
  <title>FAQ Search </title>
</head>
<body>

<%
  If query is null Then
%>

  <center>
  <h3>FAQ Search </h3>
  <form method=post action="faqsearch">
  <p>
  Search for <input type=Text size=15 maxlength=25 name="query">
  within
  <select name="tagvalue">

  <%
  -- generates the pull-down menu with the following select
    for c in (select token_text from DR$FAQ_IDX$I
              where token_type=2)
    loop
  %>
      <option value="<%= c.token_text %>"><%= c.token_text %>

  <% end loop; %>

  <option value="faq@Owner">FAQ Owner
</select>
```

```



```

```

<%
    v_query := query || ' WITHIN ' || tagvalue;

-- Text query using WITHIN for XML documents
    for doc in (select tk, xml_desc
                from faq where contains(xml_desc,v_query) >0
                )
    loop
        v_results := v_results + 1;
        if v_results = 1 then
%>
            <table border="1" cellpadding="4" cellspacing="0">
                <tr bgcolor="#6699CC">
                    <th>Title</th>
                    <th>Description</th>
                </tr>
<%
            end if; %>

            <tr bgcolor="#<%= color %>">

<%
                v_title_start := instr(doc.xml_desc,'<TITLE>');
                v_title_final := instr(doc.xml_desc,'</TITLE>');
                v_title_substr := substr(doc.xml_desc,v_title_
start+length('<TITLE>'),v_title_final - length('</TITLE>') - v_title_start+1);

                v_desc_start := instr(doc.xml_desc,'<DESCRIPTION>');
                v_desc_final := instr(doc.xml_desc,'</DESCRIPTION>');
                v_desc_substr := substr(doc.xml_desc,v_desc_
start+length('<DESCRIPTION>'),v_desc_final - length('</DESCRIPTION>') - v_desc_
start+1);
                %>

                <td>
                    <a href="showxml?id=<%= doc.tk %>"><%= v_title_substr %></a>
                </td>

<%
                v_desc_item := replace(v_desc_substr,'<','&lt;'); %>

                <td>
                    <%= v_desc_item %>
                </td>
            </tr>

```

```
<%
    if (color = 'ffffff') then /* alternate row color */
        color := 'eeeeee';
    else
        color := 'ffffff';
    end if;
end loop;
%>

</table>
</center>

<%
    if v_results = 0 then %>
        <center><h3>Found 0 records for your query</h3></center>
    <% end if; %>

<% End if;%>
<p>
<hr>
<p>
</body>
</html>
```

Frequently Asked Questions (FAQs): Oracle Text

This FAQ section is divided into the following categories:

- [Searching Attribute Values](#)
- [General Oracle Text Questions](#)
- [Searching XML Documents in CLOBs](#)

Searching Attribute Values

Can I Build Indexes on Attribute Values?

Currently Oracle Text (*intermedia* Text) has the option to create indexes based on the content of a section group. But most XML Elements are of the type of Element.

So, the only option for searching would be attribute values. Can I build indexes on attribute values?

Answer

Releases from 8.1.6 and higher allow attribute indexing. See the following site: http://otn.oracle.com/products/intermedia/htdocs/text_training_816/Samples/imt_816_techover.html#SCN

General Oracle Text Questions

Can XML Documents Be Queried Like Table Data?

I know that an intact XML document can be stored in a CLOB in ORACLE's XML solution.

1. Can XML documents stored in a CLOB/BLOB be queried like table schema?
For example:

```
[XML document stored in BLOB]...<name id="1111"><first>lee</first>
<second>jumee</second></name>...
```

Can value(lee, jumee) be queried by elements, attributes and the structure of XML document?

2. If some element or attribute is inserted/updated/deleted, must the whole document be updated? Or can insert/update/delete function as in table schema?
3. About locking, if we manage an XML document stored in a CLOB/BLOB, can nobody access the same XML document?

Answer

1. Using Oracle Text (*intermedia* Text), you can find this document with a query such as:

```
lee within first or this:jumee within second or this:1111 within name@id
```

you can combine these like this:

```
lee within first and jumee within second or this:(lee within first) within name.
```

For more information, please read the “*interMedia* Text Technical Overview” for 8.1.5 and 8.1.6 available on OTN.

2. Oracle Text (intermedia Text) indexes CLOB/BLOB, and this has no knowledge about XML specifically, so you cannot really change individual elements. You have to edit the document as a whole.
3. Just like any other CLOB, if someone is writing to the CLOB, they have it locked and nobody else can write to the CLOB. Other users can READ it, but not write. This is basic LOB behavior.

Another alternative is to decompose the XML document and store the information in relational fields. Then you could modify individual elements, have element-level simultaneous access, and so on. In this case, using something called the USER_DATASTORE, you can use PL/SQL to reconstitute the document to XML for text indexing. Then, you get text search as if it were XML, but data management as if it were relational data. Again, see interMedia Text Technical Overview for more information.

<http://otn.oracle.com/products/text>.

Can we Search Based on Structural Conditions?

Is it possible for Oracle Text (intermedia Text) to index XML such as: 2/7/1968 and then process a query such as:

Who has brown hair, that is, select name from person where hair.color = "BROWN"

Answer

Searches based on structural conditions are not yet available through Oracle Text (intermedia Text). Attribute searches are supported from Oracle8i Release 2 (8.1.6). For reference you should not put data in attributes as that will not be compliant with XML Schema when it becomes a recommendation.

How Can I Searching XML Documents and Return a Zone?

I need to store a large XML file in Oracle8i, search it, and return a specific tagged area. Using Oracle Text (intermedia Text) some of this is possible:

- I can store an XML file in a CLOB field
- I can index it with ctxsys.context
- I can create <Zones> and <Fields> to represent the Tags in my XML file. Ex. ctx_ddl.add_zone_section(xmlgroup, "dublincore", dc);
- I can search for text within a Zone or field. Ex. Select title from mytable where CONTAINS(textField, "some words WITHIN doubleness")

How do I return a zone or a field based on a text search?

Answer

Oracle Text (intermedia Text) will only return the “hits”. You will need to subsequently parse the CLOB to extract a section.

Loading XML Documents into the Database and Searching with Oracle Text

How do I insert XML documents into a database? I need to insert the XML document “as is” in column of datatype CLOB into a table.

Answer

Oracle's XML SQL Utility for Java offers a command-line utility that can be used for loading XML data. More information can be found on the XML SQL Utility at: <http://otn.oracle.com/tech/xml> and here in [Chapter 7, "XML SQL Utility \(XSU\)"](#).

You can insert the XML documents as you would any text file. There is nothing special about an XML-formatted file from a CLOB perspective.

Question 2

Oracle Text (intermedia Text) can be used to index and search XML stored in CLOBs? How can we get started?

Answer 2

Versions of Oracle Text (intermedia Text) before Oracle8i Release 2 (8.1.6) only allowed tag-based searching. The current version allows for XML structure and attribute based searching. There is documentation on how to have the index built and the SQL usage in the Oracle Text documentation.

See Also: *Oracle9i Text Reference.*

How Do I Search XML using the WITHIN Operator?

I have this xml:

```
<person>
  <name>efrat</name>
  <childrens>
    <child>
      <id>1</id>
      <name>keren</name>
```

```
    </child>
  </childrens>
</person>
```

How do I find the person who has a child name keren but not the person's name keren? Assuming I defined every tag with the `add_zone_section` that can be nested and can include themselves.

Answer

Use `selectSingleNode` or `selectNodes` with XPATH string as a parameter.eg. `selectSingleNode("//child/name[.='keren'])` Also, I recommend making `id` as an attribute instead of a tag.

Oracle Text (intermedia Text) and XML

Where can I get good samples of searching XML with Oracle Text.

Answer

See the following manuals:

- *Oracle9i Text Developer's Guide*
- *Oracle9i Text Reference*

Oracle Text (intermedia Text) and XML: Add_field_section

Can Oracle Text (intermedia Text) recognize the tags in my XML document or do I have to use the `add_field_section` command for each tag in the XML document? My XML documents have hundreds of tags. Is there an easy way to do this?

Answer

Which version of the database are you using? I believe you need to do it for 8.1.5 but not in Oracle8i Release 2(8.1.6). You can use `AUTO_SECTION_GROUP` in 8.1.6

XSQL Servlet ships with a complete (albeit simple from the *interMedia* standpoint) example of a SQL script that creates a complex XML Datagram out of Object Types, and then creates an Oracle Text (intermedia Text) index on the XML Document Fragment stored in the "Insurance Claim" type.

If you download the XSQL Servlet, and look at the file `./xsql/demo/insclaim.sql` you'll be able to see the *interMedia* stuff at the bottom of the file. One of the key

new features in *interMedia* in Oracle8i Release 2(8.1.6) is the AUTO Sectioner for XML.

Can I Do Range Searching with Oracle Text?

I have an XML document that I have stored in CLOB. I have also created the indexes on the tags using `section_group`, and so on. One of the tags is `<SALARY>` `</SALARY>` I want to write an SQL statement so as to select all the records that have salary lets say `> 5000`. How do I do this? I cannot use `WITHIN` operator. I want to interpret the value present in this tag as a number. This could be floating point number also since this is salary.

Answer

You cannot do this in Oracle Text. Range search is not really a text operation. The best solution is to use the other Oracle XML parsing utilities to extract the salary into a `NUMBER` field -- then you can use Oracle Text (*intermedia* Text) for text searching, and normal SQL operators for the more structured fields, and achieve the same results.

Can Oracle Text Do Section Extraction?

We are storing all our documents in XML format in a CLOB. Are there utilities available in Oracle perhaps *interMedia* to retrieve the contents a field at a time, that is given a field name, get the text between tags, as opposed to retrieving the whole document and traversing the structure?

Answer

interMedia does not do section extraction. See the XML SQL Utility for this in [Chapter 7, "XML SQL Utility \(XSU\)"](#).

Can I Create a Text Index on Three Columns?

I have created a view based on 7-8 tables and it has columns like, `custordnumber`, `product_dscr`, `qty`, `prdid`, `shipdate`, `ship_status`, and so on. I need to create an Oracle Text index on the three columns:

- `custordnumber`
- `product_dsc`
- `ship_status`

Is there a way to create a Text index on these columns?

Answer

The short answer is yes. You have two options:

1. Use the USER_DATASTORE object to create a concatenated field on the fly during indexing;
2. Concatenate your fields and store them in an extra CLOB field in one of your tables. Then create the index on the CLOB field. If you're using Oracle8i Release 2(8.1.6) or higher, then you also have the option of placing XML tags around each field prior to concatenation. This gives you the capability of searching WITHIN each field.

How Fast is Oracle at Indexing Text and Can I Just Enable Boolean Searches?

We are using MySQL to do partial indexing of 9 million Web pages a day. We are running on a 4-processor Sparc 420 and unable to do full text indexing. Can Oracle8i or Oracle9i do this?

We are not interested in transactional integrity, applying any special filters to the text pages, or in doing any other searching other than straight boolean word searches (no scoring, no stemming, no fuzzy searches, no proximity searches, and so on).

I have are the following questions:

- Will Oracle be any faster at indexing text than MySQL?
- If so, is there a way to disable all the features of text indexing except for boolean word searches?

Answer

Yes. Oracle Text (*interMedia Text*) can create a full-text index on 9 million web pages - and pretty quickly. In a benchmark on a large Sun box, we indexed 100Gig of web pages (about 15 million) in 7 hours. We can also do partial indexing via regular DML or via partitioning.

You can do "indexing light" to some extent - you can disable theme indexing, you do not need to filter documents if they are already ASCII/HTML/XML, and most common expansions - fuzzy, stemming, proximity - are done at query time.

How Can We Index XML Documents in Different Languages?

We know that Oracle 8i Release 2 (8.1.6) allows multiple language records to be stored in the same table (and column) and interMedia handles the index appropriately based on the language setting for each row (using the multi-lexer feature).

Currently we use one CLOB column in the table and it is indexed using Oracle Text. The column content is in XML (tagged) format and we use fields, sections and zone functions for indexing and search. This works perfectly as we only have a single language data in the database (and we have different database for different languages and sites) and we are currently using Oracle 8i Release 1 (8.1.5) so we have NLS_LANG appropriately set for indexing and searches work correctly for individual languages.

However, we now have to store multi-lingual data in the same table (and column). Individual data elements may also be in different languages. For example, this is a record for a book that has a French title but Spanish authors. At present we have all this information in the CLOB column separated by fields/sections. My questions are:

1. I presume there is no way to specify language for individual sections within an index in Oracle 8i Release 2 (8.1.6) Is this correct?
2. We could separate out all the fields that could potentially be in different language into different columns in the same table and then have a corresponding language column for each of those columns and use the multi-lexer functionality to build separate indexes. Is this assumption correct or recommended?
3. If we do as described above, then we need to have multiple CONTAINS clauses when searching across columns, which can adversely affect performance.
4. How best we can approach this issue?

Answer

1. Correct.
- 2) - 3) You have correctly identified the potential problem.

Searching XML Documents in CLOBs

How Do I Search CLOBs Using Oracle Text?

How would I define *interMedia* parameters so that I would be able to search my CLOB column for records that contained “aorta” and “damage”. For example using the following XML (DTD implied):

```
WellKnownFileName.gif echocardiogram aorta
```

This is an image of the vessel damage. It would be nice to see a simple (or complicated) example of an XML *interMedia* implementation. I assume there is no need to setup the ZONE or FIELDS.....Is this the case?

Answer

If you save an XML Document fragment in a CLOB, and enable an Oracle Text (*intermedia* Text) XML index on it, then you can do an SQL query which uses the CONTAINS() operator as the following query does:

Assume you have a document like an insurance claim:

```
77804
1999-01-01 00:00:00.0          8895          1044
Paul          Astoria
123 Cherry Lane      SF          CA          94132
1999-01-05 00:00:00.0          7600          JCOX
It was because of Faulty Brakes
```

If you store the content as a document fragment in a CLOB, then you can do a query like the following (assuming everything else you store in relational tables):

```
REM Select the SUM of the amounts of
REM all settlement payments approved by "JCOX"
REM for claims whose relates to Brakes.
select sum(n.amount) as TotalApprovedAmount
  from insurance_claim_view v, TABLE(v.settlements) n
 where n.approver = 'JCOX'
       and contains(damageReport, 'Brakes within Cause') >
```

How Can I Search Different XML Documents Stored in CLOBs With Different DTDs?

If I store XML in CLOBs and use the DOM or SAX to reparse the XML later as needed. How can I search this document repository? Oracle Text (*intermedia* Text) seems ideal. Do you have an example of setting this up using *intermedia* in

Oracle8i, demonstrating how to define the XML_SECTION_GROUP and where to use a ZONE as opposed to a FIELD, and so on? For example:

How would I define Intermedia parameters so that I would be able to search my CLOB column for records that had the “aorta” and “damage” in the using the following XML (DTD implied) WellKnownFileName.gif echo cardiogram aorta This is an image of the vessel damage.

Answer

Oracle8i Release 2 (8.1.6) and higher allow searching within attribute text. That's something like: state within book@author. Oracle now offers attribute value sensitive search, more like the following:

state within book[@author = “Eric”]:

```
begin ctx_ddl.create_section_group('mygrp','basic_section_group');
  ctx_ddl.add_field_section('mygrp','keyword','keyword');
  ctx_ddl.add_field_section('mygrp','caption','caption');
end;
create index myidx on mytab(mytxtcolumn) indextype is ctxsys.contextparameters
('section group mygrp');
select * from mytab where contains(mytxtcolumn, 'aorta within keyword')>0;
options:
```

- Use XML section group instead of basic section group if your tags have attributes or you need case-sensitive tag detection
- Use zone sections instead of field sections if your sections overlap, or if you need to distinguish between instances. For instance, keywords. If keywords is a field section, then (aorta and echo cardiogram) within keywords finds the document. If it is a zone section, then it does not, because they are not in the SAME instance of keywords.

It is not so clear. It looks to me like this example is trying to find instances of elements containing “damage” that have a sibling element containing “aorta” within the same record. It's not clear what exactly he means by “record”.

If each record equates to the in his example, and there can be multiple records in a single XML LOB, than I don't see how you could do this search with *interMedia*.

If there is only one per CLOB/row, than perhaps you could find it by ANDing two context element queries. But that would still be a sloppy sort of xml search relying on some expected limitations of the situation more so than the structural composition actually called for.

Storing an XML Document in CLOB: Using Oracle Text (intermedia Text)

I need to store XML files (that are present on the file system as of now) into the database. I want to store the whole document. What I mean is that I do not want to break the document as per the tags and then store the info in separate tables/fields. Rather I want to have a universal table, that I can use to store different XML documents. I think internally it will be stored in a CLOB type of field in my case. My XML files will always contain ASCII data.

Can this be done using *interMedia*. Should we be using Oracle Text (intermedia Text) or interMedia Annotator for this? I downloaded Annotator from OTN, but I could not store XML document in the database.

I am trying to store XML document into CLOB column. Basically I have one table with the following definition:

```
CREATE TABLE xml_store_testing
(
    xml_doc_id NUMBER,
    xml_doc     CLOB )
```

I want to store my XML document in xml_doc field.

I have written another PL/SQL procedure shown below, to read the contents of the XML Document. The XML document is available on the file system. XML document contains just ASCII data - no binary data.

```
CREATE OR REPLACE PROCEDURE FileExec
(
    p_Directory      IN VARCHAR2,
    p_FileName       IN VARCHAR2)
AS
    v_CLOBLocator    CLOB;
    v_FileLocator     BFILE;
BEGIN
    SELECT  xml_doc
    INTO    v_CLOBLocator
    FROM    xml_store_testing
    WHERE   xml_doc_id = 1
    FOR    UPDATE;
    v_FileLocator := BFILENAME(p_Directory, p_FileName);
    DBMS_LOB.FILEOPEN(v_FileLocator, DBMS_LOB.FILE_READONLY);
    dbms_output.put_line(to_char(DBMS_LOB.GETLENGTH(v_FileLocator)));
    DBMS_LOB.LOADFROMFILE(v_CLOBLocator, v_FileLocator,
        DBMS_LOB.GETLENGTH(v_FileLocator));
    DBMS_LOB.FILECLOSE(v_FileLocator);
END FileExec;
```

Answer

Put the XML documents into your CLOB column, then add an Oracle Text (intermedia Text) index on it using the XML section-group. See the documentation and overview material at <http://otn.oracle.com/products/intermedia>.

Question 2

When I execute this procedure, it executes successfully. But when I select from the table I see unknown characters in the table in CLOB field. Could this be because of the reason of the character set difference between operating system (where XML file resides) and database (where CLOB data resides).

Answer 2

Yes. If the character sets are different then you probably have to pass the data through `UTL_RAW.CONVERT` to do a character set conversion before writing to the CLOB.

Can We Only Insert Structured When The Table is Created?

We need to insert data in the Database from an XML file. Currently we only can insert structured data with the table already created. Is this true?

We are working in a law project where we need to store laws that have structured data and unstructured data, and then search the data using Oracle Text (*interMedia Text*). Can we insert unstructured data too? Or do we need to develop a custom application to do it? Then if we have the data stored with some structured parts and some unstructured parts, can we use Oracle Text to search it? If we stored the unstructured part in a CLOB, and the CLOB has tags, how can we search only data in an specific tag?

Answer

Consider using *iFS* which allows you to break up a document storing it across tables and in a LOB. Oracle Text can perform data searches with tags and is knowledgeable about the hierarchical XML structure. From Oracle8i Release 2 (8.1.6), Oracle Text (intermedia Text) has this capability along with name/value pair attribute searches.

Question 2

Hence, this document breaking is not possible in these cases if I don't create a custom development? Although *interMedia* does not understand hierarchical XML structure, can I do something like this?

```
<report>
  <day>yesterday</day> there was a disaster <cause>hurricane</cause>
</report>
```

Indexing with Oracle Text I would like to search LOBs where cause was hurricane. Is this possible?

Answer 2

You can perform that level of searching with the current release of Oracle Text (intermedia Text). Currently to break a document up you have to use the XML Parser with XSLT to create a stylesheet that transforms the XML into DDL. *iFS* gives you a higher level interface.

Another technique is to use a JDBC program to insert the text of the document or document fragment into a CLOB or LONG column, then do the searching using the CONTAINS() operator after setting up the indexes.

Part III

Data Exchange Using XML

Part III includes a description of Oracle Advanced Queuing (AQ), the new AQ iDAP feature, XML queues, XML message transformation, and how AQ and XML can be used in B2B messaging applications.

Part III contains the following chapters:

- [Chapter 9, "Exchanging XML Data Using Oracle AQ"](#)

Exchanging XML Data Using Oracle AQ

This chapter contains the following sections:

- What is AQ?
- How do AQ and XML Complement Each Other?
- Internet-Data-Access-Presentation (IDAP)
 - IDAP Architecture
 - IDAP Message Body is an AQ XML Document
 - IDAP Client Requests for Enqueue
 - IDAP Client Requests for Dequeue
 - IDAP Client Requests for Registration
 - IDAP Server Response to Enqueue
 - Server Response to a Dequeue Request
 - Server Response to a Register Request
 - Notification
 - IDAP and AQ XML Schemas
- AQXMLServlet
- XMLType Queues
- AQ XML Message Format Transformation
- Frequently Asked Questions (FAQs): XML and Advanced Queuing

What is AQ?

Oracle Advanced Queuing (AQ) provides database integrated message queuing functionality. AQ:

- Enables and manages asynchronous communication of two or more applications using messages.
- Supports point-to-point and publish/subscribe communication models.

Integration of message queuing with Oracle database brings the integrity, reliability, recoverability, scalability, performance, and security features of Oracle to message queuing. Integration with Oracle also facilitates the extraction of intelligence from message flows.

How do AQ and XML Complement Each Other?

XML has emerged as a standard format for business communications. XML is being used not only to represent data communicated between business applications, but also, the business logic that is encapsulated in the XML.

In Oracle, AQ supports native XML messages and also allows AQ operations to be defined in the XML-based Internet-Data-Access-Presentation (IDAP) format. IDAP, an extensible message invocation protocol, is built on Internet standards, using HTTP and email protocols as the transport mechanism, and XML as the language for data presentation. Clients can access AQ using this.

See "[Internet-Data-Access-Presentation \(IDAP\)](#)" on page 9-6.

AQ and XML Message Payloads

[Figure 9-1](#) shows an Oracle database using AQ to communicate with three applications, with XML as the message payload. The general tasks performed by AQ in this scenario are:

- Message flow using subscription rules
- Message management
- Extracting business intelligence from messages
- Message transformation

This is an *intra*- and *inter*-business scenario where XML messages are passed asynchronously among applications using AQ.

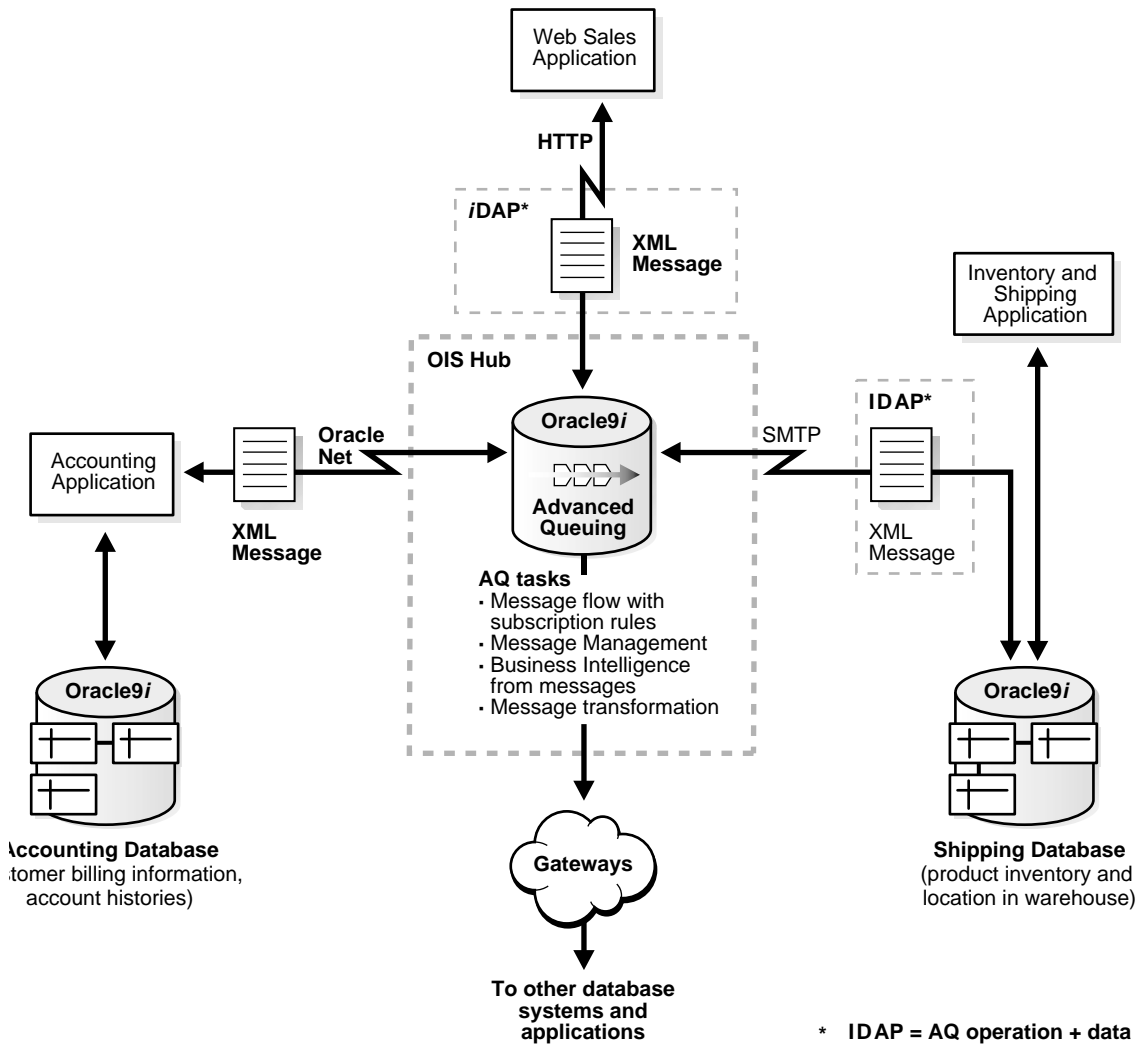
- Intra-business. Typical examples of this kind of scenario include sales order fulfillment and supply-chain management.
- Inter-business processes. Here multiple integration hubs can communicate over the Internet backplane. Examples of inter-business scenarios include travel reservations, coordination between manufacturers and suppliers, transferring of funds between banks, and insurance claims settlements, among others.

Oracle uses this in its enterprise application integration products. XML messages are sent from applications to an AQ hub, here shown as an OIS hub. This serves as a “message server” for any application that wants the message. Through this hub and spoke architecture, XML messages can be communicated asynchronously to multiple loosely-coupled receiving applications.

Figure 9-1 shows XML payload messages transported using AQ in the following ways:

- Web-based application that uses an AQ operation over an HTTP connection using IDAP
- Accounting application that uses AQ to propagate an XML message over a Net* connection.
- Shipping and inventory application that uses AQ to propagate an IDAP/XML message directly to the database over HTTP.

Figure 9-1 Advanced Queueing and XML Message Payloads



AQ Enables Hub-and-Spoke Architecture for Application Integration

A critical challenge facing enterprises today is application integration. Application integration involves getting multiple departmental applications to cooperate, coordinate, and synchronize in order to execute complex business transactions.

Advanced Queuing enables hub-and-spoke architecture for application integration. It makes integrated solution easy to manage, easy to configure, and easy to modify with changing business needs.

Messages Can be Retained for Auditing, Tracking, and Mining

Message management provided by AQ is not only used to manage the flow of messages between different applications, but also, messages can be retained for future auditing and tracking, and extracting business intelligence.

Viewing Message Content With SQL Views

AQ also provides SQL views to look at the messages. These SQL views can be used to analyze the past, current, and future trends in the system.

Advantages of Using AQ

AQ provides the flexibility of *configuring communication* between different applications.

Internet-Data-Access-Presentation (IDAP)

You can now perform AQ operations over the Internet by using Internet Data Access Presentation (IDAP). IDAP defines the message structure using XML. IDAP-structured message is sent over the Internet using transport protocols such as HTTP or SMTP.

XML and the IDAP Interface

The Internet Data Access Presentation (IDAP) uses the Content-Type of `text/xml` to specify the body of the request containing an XML-encoded method request. XML provides the presentation for IDAP request and response messages as follows:

- All protocol tags are scoped to the IDAP namespace.
- The sender includes namespaces in IDAP elements and attributes.
- The receiver processes IDAP messages that have correct namespaces; for requests with incorrect namespaces, the receiver returns an invalid request error.
- The receiver processes IDAP messages without namespaces as though they had the correct namespaces, if the context is valid.
- The IDAP namespace has the value `http://ns.oracle.com/AQ/schemas/envelope`
- An XML document forming the request of an IDAP invocation does not require the use of an XML DTD or a schema.

See Also: *Oracle8i Application Developer's Guide - Advanced Queuing*

IDAP Architecture

Figure 9–2 shows the following components needed to send HTTP messages:

- Client program which sends XML messages, that conform to IDAP format, to the AQ Servlet. This can be any HTTP client, such as, Web browsers.
- The Webserver or ServletRunner which hosts the AQ servlet that can interpret the incoming XML messages, for example, Apache/Jserv or Tomcat
- Oracle. The AQ servlet connects to this server to perform operations in your queues.

The AQ client program sends XML messages (conforming to IDAP) to the AQ servlet. Any HTTP client, for example Web browsers, can be used. The Web server/ServletRunner hosting the AQ servlet interprets the incoming XML messages. Examples include Apache/Jserv or Tomcat. The AQ servlet connects to the Oracle database server and performs operations on the users' queues.

Figure 9–2 IDAP Architecture for Performing AQ Operations Using HTTP

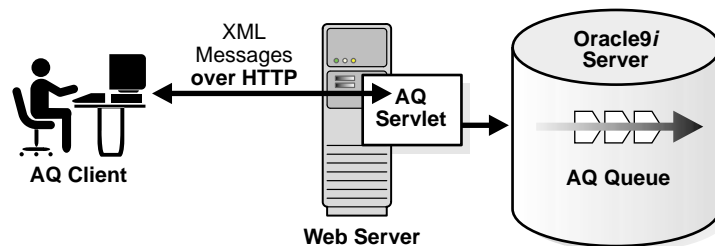
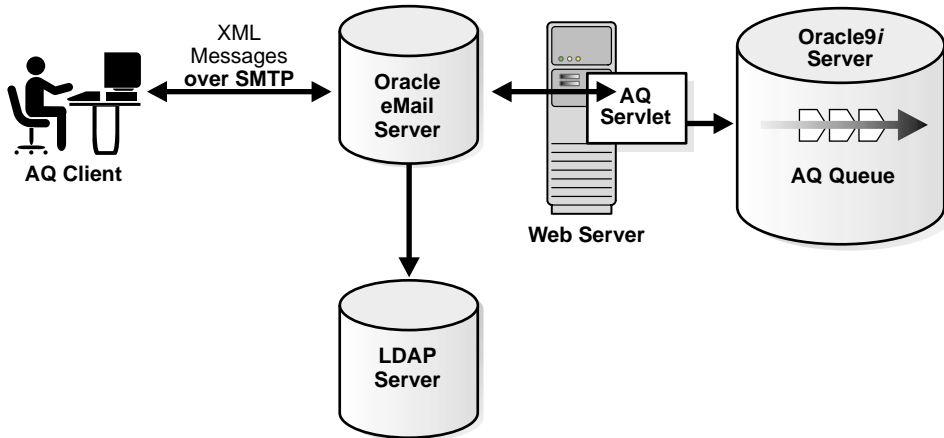


Figure 9–3 shows IDAP architecture when using SMTP. For SMTP, you will need the following two additional components:

- An Email Server
- An LDAP Server

The Email server verifies client signatures using certificates stored in LDAP and then routes the request to the AQ servlet.

Figure 9–3 IDAP Architecture for Performing AQ Operations Using SMTP



IDAP Method Invocation

A method invocation is performed by creating the request header and body and processing the returned response header and body. The request and response headers can consist of standard transport protocol-specific and extended headers.

HTTP Headers

The `POST` method within the HTTP request header performs the IDAP method invocation. The request should include the header `IDAPMethodName`, whose value indicates the method to be invoked on the target. The value consists of a URI followed by a "#", followed by a method name (which must not include the "#" character), as follows:

```
IDAPMethodName: http://ns.oracle.com/AQ/schemas/access#AQXmlSend
```

The URI used for the interface must match the implied or specified namespace qualification of the method name element in the `IDAP:Body` part of the payload.

SMTP Headers

In the case of SMTP (email), the method invocation can be done by the filter interface of the email server, which invokes a Java method with the email-message-body as argument. This results in remote invocation of the `POST` method on the AQ servlet. The response is emailed directly to the recipient

specified in the reply of the message. The response header can contain SMTP-protocol-related headers also.

IDAP Message Structure

IDAP structures a message request or response as follows:

- IDAP envelope (the root or top element in an XML tree)
- IDAP header (first element under the root)
- IDAP body (the AQ XML document)

The IDAP Envelope

The tag of this root element is `IDAP:Envelope`. IDAP defines a global attribute `IDAP:encodingStyle` that indicates serialization rules used instead of those described by the IDAP specification. This attribute may appear on any element and is scoped to that element and all child elements not themselves containing such an attribute. Omitting `IDAP:encodingStyle` means that type specification has been followed (unless overridden by a parent element).

The IDAP envelope also contains namespace declarations and additional attributes, provided they are namespace-qualified. Additional namespace-qualified subelements can follow the body.

IDAP Headers

The tag of this first element under the root is `IDAP:Header`. An IDAP header passes necessary information, such as the transaction ID, with the request. The header is encoded as a child of the `IDAP:Envelope` XML element. Headers are identified by the name element and are namespace-qualified. A header entry is encoded as an embedded element.

The IDAP Body

The IDAP body, tagged `IDAP:Body`, contains a first subelement whose name is the method name. This method request element contains elements for each input and output parameter. The element names are the parameter names. The body also contains `IDAP:Fault`, indicating information about an error.

For performing AQ operations, the IDAP body must contain an AQ XML document. The AQ XML document has the namespace `http://ns.oracle.com/AQ/schemas/access`

IDAP Method Invocation Body: “IDAP Payload”

IDAP method invocation consists of a method request and optionally a method response. The IDAP method request and method response are HTTP request and response, respectively, whose content is an XML document that consists of the root and mandatory body elements. This XML document is referred to as *IDAP payload* in the rest of this chapter.

The IDAP payload is defined as follows:

- IDAP root element is the top element in the XML tree.
- IDAP payload headers contain additional information that must travel with the request.
- The method request is represented as an XML element with additional elements for parameters. It is the first child of the IDAP:Body element. This request can be one of the AQ XML client requests described in the next section
- The response is the return value or error/exception that is passed back to the client. The encoding rules are as follows:

Requests: Outcomes at the Receiving Site

At the receiving site, a request can have one of the following four outcomes:

- a. The HTTP infrastructure on the receiving site was able to receive and process the request. The HTTP infrastructure passes the headers and body to the IDAP infrastructure.
- b. The HTTP infrastructure on the receiving site could not receive and process the request. The result is an HTTP response containing an HTTP error in the status field and no XML body.
- c. The IDAP infrastructure on the receiving site was able to decode the input parameters, dispatch to an appropriate server indicated by the server address, and invoke an application-level function corresponding semantically to the method indicated in the method request. The result of the method request consists of a response or error.
- d. IDAP infrastructure on the receiving site could not decode the input parameters, dispatch to an appropriate server indicated by the server address, and invoke an application-level function corresponding semantically to the interface or method indicated in the method request. The result of the method is a error indicating a error that prevented the dispatching infrastructure on the receiving side from successful completion.

In (c) and (d), additional message headers may, for extensibility, again be present in the request results.

Results from a Method Request

The results of the request are provided in a request-response format. The HTTP response must be of Content-Type “text/xml”.

An IDAP result indicates success. An error indicates failure. The method response will never contain both a result and an error. The different types of responses and errors are described in the next section

IDAP Message Body is an AQ XML Document

The body of an IDAP message is an AQ XML document, which represents:

- Client requests for enqueue, dequeue, and registration
- Server responses to client requests for enqueue, dequeue, and registration
- Notifications from the server to the client

Note: AQ Internet Access is supported only for 8.1-style queues. 8.0-style queues cannot be accessed using IDAP.

IDAP Client Requests for Enqueue

Client requests for enqueue—`SEND` and `PUBLISH` requests—use the following methods:

- `AQXmlSend`—to enqueue to a single-consumer queue
- `AQXmlPublish`—to enqueue to multiconsumer queues/topics

`AQXmlSend` and `AQXmlPublish` take the arguments and argument attributes shown in [Table 9-1](#). Required arguments are shown in bold.

Table 9–1 IDAP Client Requests for Enqueue—Arguments and Attributes for AQXmiSend and AQXmiPublish

Argument	Attribute
<p>producer_options</p>	<p>destination—specify the queue/topic to which messages are to be sent. The destination element has an attribute <code>lookup_type</code> which determines how the destination element value is interpreted</p> <ul style="list-style-type: none"> ▪ DATABASE (default)—destination is interpreted as <code>schema.queue_name</code> ▪ LDAP—the LDAP server is used to resolve the destination <p>visibility</p> <ul style="list-style-type: none"> ▪ ON_COMMIT—The enqueue is part of the current transaction. The operation is complete when the transaction commits. This is the default case. ▪ IMMEDIATE—effects of the enqueue are visible immediately after the request is completed. The enqueue is not part of the current transaction. The operation constitutes a transaction on its own. <p>transformation—the PL/SQL transformation to be invoked before the message is enqueued</p>
<p>message_set—contains one or more messages.</p> <ul style="list-style-type: none"> ▪ message_header 	<p>Each message consists of a <code>message_header</code> and <code>message_payload</code></p> <p>message_id—unique identifier of the message, supplied during dequeue</p> <p>correlation—correlation identifier of the message</p> <p>expiration—duration in seconds that a message is available for dequeuing. This parameter is an offset from the delay. By default messages never expire. If the message is not dequeued before it expires, then it is moved to the exception queue in the EXPIRED state</p> <p>delay—duration in seconds after which a message is available for processing</p> <p>priority—the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.</p> <p>sender_id—the application-specified identifier</p> <ul style="list-style-type: none"> ▪ agent_name, address, protocol ▪ agent_alias—if specified, resolves to a name, address, protocol using LDAP

Table 9–1 IDAP Client Requests for Enqueue—Arguments and Attributes for AQXmlSend and AQXmlPublish

Argument	Attribute
	<p>recipient_list—overrides the default subscriber list; lookup_type defines if the recipients are specified or looked up in LDAP</p> <ul style="list-style-type: none"> ▪ agent_name, address, protocol ▪ agent_alias—if specified, resolves to a name, address, protocol using LDAP
	<p>message_state—the state of the message is filled in automatically during dequeue</p> <p>0: The message is ready to be processed. 1: The message delay has not yet been reached. 2: The message has been processed and is retained. 3: The message has been moved to the exception queue.</p> <p>exception_queue—in case of exceptions the name of the queue to which the message is moved if it cannot be processed successfully. Messages are moved in two cases: The number of unsuccessful dequeue attempts has exceeded <i>max_retries</i> or the message has expired. All messages in the exception queue are in the EXPIRED state.</p> <p>The default is the exception queue associated with the queue table. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert file. If the default exception queue is used, then the parameter returns a NULL value at dequeue time.</p> <ul style="list-style-type: none"> ▪ message_payload this can have different sub-elements based on the payload type of the destination queue/topic. The different payload types are described in the next section
AQXmlCommit	this is an empty element—if specified, the user transaction is committed at the end of the request

Message Payloads

AQ supports messages of the following types:

- [RAW Queues](#)
- [Oracle Object \(ADT\) Type Queues](#)
- [Java Message Service \(JMS\) Type Queues/Topics](#)

All these types of queues can be accessed using IDAP. If the queue holds messages in RAW, Oracle object, or JMS format, XML payloads are transformed to the

appropriate internal format during enqueue and stored in the queue. During dequeue, when messages are obtained from queues containing messages in any of the above formats, they are converted to XML before being sent to the client.

The message payload type depends on the type of the queue on which the operation is being performed. A discussion of the queue types follows:

RAW Queues

The contents of RAW queues are raw bytes. The user must supply the hex representation of the message payload in the XML message. For example, `<raw>023f4523</raw>`.

Oracle Object (ADT) Type Queues

For ADT queues that are not JMS queues (that is, they are not type `AQ$_JMS_*`), the type of the payload depends on the type specified while creating the queue table that holds the queue. The XML specified here must map to the SQL type of the payload for the queue table.

See Also:

- [Chapter 5, "Database Support for XML"](#)
- [Chapter 7, "XML SQL Utility \(XSU\)"](#)

for more details on mapping SQL types to XML.

ADT Type Queues Example Assume the queue is defined to be of type `EMP_TYP`, which has the following structure:

```
create or replace type emp_typ as object (  
    empno NUMBER(4),  
    ename VARCHAR2(10),  
    job VARCHAR2(9),  
    mgr NUMBER(4),  
    hiredate DATE,  
    sal NUMBER(7,2),  
    comm NUMBER(7,2)  
    deptno NUMBER(2));
```

The corresponding XML representation is:

```
<EMP_TYP>  
  <EMPNO>1111</EMPNO>  
  <ENAME>Mary</ENAME>
```

```

<MGR>5000</MGR>
<HIREDATE>1996-01-01 0:0:0</HIREDATE>
<SAL>10000</SAL>
<COMM>100.12</COMM>
<DEPTNO>60</DEPTNO>
</EMP_TYP>

```

Java Message Service (JMS) Type Queues/Topics

For queues with JMS types (that is, those with payloads of type `AQ$_JMS_*`), there are four different XML elements, depending on the JMS type. Hence, IDAP supports queues/topics with the following JMS types:

- `TextMessage`
- `MapMessage`
- `BytesMessage`
- `ObjectMessage`

Note: JMS queues with payload type `StreamMessage` are not supported through IDAP.

[Table 9–2](#) lists the JMS types and XML components. The distinct XML element for each JMS type is shown in its respective column. Required elements are shown in bold.

Table 9–2 *JMS Types and XML Components*

jms_text_message	jms_map_message	jms_bytes_message	jms_object_message
Used for queues/topics with payload type:	-	-	-
AQ\$_JMS_TEXT_MESSAGE	AQ\$_JMS_MAP_MESSAGE	AQ\$_JMS_BYTES_MESSAGE	AQ\$_JMS_OBJECT_MESSAGE

Table 9–2 JMS Types and XML Components (Cont.)

jms_text_message	jms_map_message	jms_bytes_message	jms_object_message
oracle_jms_ properties	-	-	-
user_properties	-	-	-
text_data —string representing the text payload	map_data —set of name-value pairs called items, consisting of: <ul style="list-style-type: none"> ■ name ■ int_value or string_value or long_value or double_value or boolean_value or float_value or short_value or byte_value 	bytes_data —hex representation of the payload bytes	ser_object_data —hex representation of the serialized object

All JMS messages consist of the following common elements:

- oracle_jms_properties, which consists of
 - type—type of the message
 - reply_to—consists of an agent_name, address, and protocol
 - userid—supplied by AQ; client cannot specify
 - appid—application identifier
 - groupid—group identifier
 - group_sequence—sequence within the group identified by group_id
 - timestamp—the time the message was sent, which cannot be specified during enqueue. It is automatically populated in a message that is dequeued.
 - recv_timestamp—the time the message was received

- `user_properties`—in addition to the above predefined properties, users can also specify their own message properties as name-value pairs. The `user_properties` consists of a list of property elements. Each property is a name-value pair consisting of the following:
 - `name`—property name
 - `int_value`—integer property value or
`string_value`—string property value or
`long_value`—long property value or
`double_value`—double property value or
`boolean_value`—boolean property value or
`float_value`—float property value or
`short_value`—short property value or
`byte_value`—byte property value or

The following examples show enqueue requests using the different message and queue types.

IDAP Enqueue Request Example1 — ADT Message to a Single-Consumer Queue

The queue `QS.NEW_ORDER_QUEUE` has a payload of type `ORDER_TYP`.

```
<?xml version="1.0"?>
  <Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
    <Body>
      <AQXmlSend xmlns = "http://ns.oracle.com/AQ/schemas/access">
        <producer_options>
          <destination>QS.NEW_ORDERS_QUEUE</destination>
        </producer_options>

        <message_set>
          <message_count>1</message_count>

          <message>
            <message_number>1</message_number>

            <message_header>
              <correlation>ORDER1</correlation>
              <sender_id>
                <agent_name>scott</agent_name>
```

```
</sender_id>
</message_header>

<message_payload>

<ORDER_TYP>
  <ORDERNO>100</ORDERNO>
  <STATUS>NEW</STATUS>
  <ORDERTYPE>URGENT</ORDERTYPE>
  <ORDERREGION>EAST</ORDERREGION>
  <CUSTOMER>
    <CUSTINO>1001233</CUSTINO>
    <CUSTID>MA123455623212</CUSTID>
    <NAME>AMERICAN EXPRESS</NAME>
    <STREET>EXPRESS STREET</STREET>
    <CITY>REDWOOD CITY</CITY>
    <STATE>CA</STATE>
    <ZIP>94065</ZIP>
    <COUNTRY>USA</COUNTRY>
  </CUSTOMER>
  <PAYMENTMETHOD>CREDIT</PAYMENTMETHOD>
  <ITEMS>
    <ITEMS_ITEM>
      <QUANTITY>10</QUANTITY>
      <ITEM>
        <TITLE>Perl</TITLE>
        <AUTHORS>Randal</AUTHORS>
        <ISBN>ISBN20200</ISBN>
        <PRICE>19</PRICE>
      </ITEM>
      <SUBTOTAL>190</SUBTOTAL>
    </ITEMS_ITEM>
    <ITEMS_ITEM>
      <QUANTITY>20</QUANTITY>
      <ITEM>
        <TITLE>XML</TITLE>
        <AUTHORS>Micheal</AUTHORS>
        <ISBN>ISBN20212</ISBN>
        <PRICE>59</PRICE>
      </ITEM>
      <SUBTOTAL>590</SUBTOTAL>
    </ITEMS_ITEM>
  </ITEMS>
  <CCNUMBER>NUMBER01</CCNUMBER>
  <ORDER_DATE>2000-08-23 0:0:0</ORDER_DATE>
```



```

        </ORDER_TYP>
    </message_payload>
</message>
</message_set>
</AQXmlSend>
</Body>
</Envelope>

```

IDAP Enqueue Request Example 2 — Message to a Multiconsumer Queue

The multiconsumer queue `AQUSER.EMP_TOPIC` has a payload of type `EMP_TYP`. `EMP_TYP` has the following structure:

```

create or replace type emp_typ as object (
    empno NUMBER(4),
    ename VARCHAR2(10),
    job VARCHAR2(9),
    mgr NUMBER(4),
    hiredate DATE,
    sal NUMBER(7,2),
    comm NUMBER(7,2)
    deptno NUMBER(2));

```

A `PUBLISH` request has the following format:

```

<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlPublish xmlns = "http://ns.oracle.com/AQ/schemas/access">
      <producer_options>
        <destination>AQUSER.EMP_TOPIC</destination>
      </producer_options>

      <message_set>
        <message_count>1</message_count>

        <message>
          <message_number>1</message_number>

          <message_header>
            <correlation>NEWEMP</correlation>
            <sender_id>
              <agent_name>scott</agent_name>
            </sender_id>
          </message_header>

```

```
<message_payload>
  <EMP_TYP>
    <EMPNO>1111</EMPNO>
    <ENAME>Mary</ENAME>
    <MGR>5000</MGR>
    <HIREDATE>1996-01-01 0:0:0</HIREDATE>
    <SAL>10000</SAL>
    <COMM>100.12</COMM>
    <DEPTNO>60</DEPTNO>
  </EMP_TYP>
</message_payload>
</message>
</message_set>
</AQXmlPublish>
</Body>
</Envelope>
```

IDAP Enqueue Request Example 3 — Sending a Message to a JMS Queue

The JMS queue `AQUSER.JMS_TEXTQ` has payload type JMS Text message (`SYS.AQ$JMS_TEXT_MESSAGE`). The send request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
  <Body>

    <AQXmlSend xmlns="http://ns.oracle.com/AQ/schemas/access">
      <producer_options>
        <destination>AQUSER.JMS_TEXTQ</destination>
      </producer_options>

      <message_set>
        <message_count>1</message_count>

        <message>
          <message_number>1</message_number>

          <message_header>
            <correlation>text_msg</correlation>
            <sender_id>
              <agent_name>john</agent_name>
            </sender_id>
          </message_header>
```

```

<message_payload>

  <jms_text_message>
    <oracle_jms_properties>
      <appid>AQProduct</appid>
      <groupid>AQ</groupid>
    </oracle_jms_properties>

    <user_properties>
      <property>
        <name>Country</name>
        <string_value>USA</string_value>
      </property>
      <property>
        <name>State</name>
        <string_value>California</string_value>
      </property>
    </user_properties>

    <text_data>All things bright and beautiful</text_data>
  </jms_text_message>
</message_payload>
</message>
</message_set>
</AQXmlSend>
</Body>
</Envelope>

```

IDAP Enqueue Request Example 4 — Sending/Publishing and Committing

```

<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlPublish xmlns = "http://ns.oracle.com/AQ/schemas/access">
      <producer_options>
        <destination>AQUSER.EMP_TOPIC</destination>
      </producer_options>

      <message_set>
        <message_count>1</message_count>

        <message>
          <message_number>1</message_number>

```

```
<message_header>
<correlation>NEWEMP</correlation>
<sender_id>
  <agent_name>scott</agent_name>
</sender_id>
</message_header>

<message_payload>
  <EMP_TYP>
    <EMPNO>1111</EMPNO>
    <ENAME>Mary</ENAME>
    <MGR>5000</MGR>
    <HIREDATE>1996-01-01 0:0:0</HIREDATE>
    <SAL>10000</SAL>
    <COMM>100.12</COMM>
    <DEPTNO>60</DEPTNO>
  </EMP_TYP>
</message_payload>
</message>
</message_set>

<AQXmlCommit/>

</AQXmlPublish>
</Body>
</Envelope>
```

IDAP Client Requests for Dequeue

Client requests for dequeue use the `AQXmlReceive` method. [Table 9-3](#) lists `AQXmlReceive` method's arguments and argument attributes. Required arguments are shown in bold.

Table 9–3 IDAP Client Requests for Dequeue—Arguments and Attributes for AQXMLReceive

Argument	Attribute
consumer_options	<p>destination—specify the queue/topic from which messages are to be received. The destination element has an attribute <code>lookup_type</code> which determines how the destination element value is interpreted</p> <ul style="list-style-type: none"> ■ <code>DATABASE (default)</code>—destination is interpreted as <code>schema.queue_name</code> ■ <code>LDAP</code>—the LDAP server is used to resolve the destination <p>consumer_name—Name of the consumer. Only those messages matching the consumer name are accessed. If a queue is not set up for multiple consumers, then this field should not be specified</p> <p>wait_time—the time (in seconds) to wait if there is currently no message available which matches the search criteria</p> <p>selector—criteria used to select the message, specified as one of:</p> <ul style="list-style-type: none"> ■ <code>correlation</code>—the correlation identifier of the message to be dequeued. ■ <code>message_id</code>— the message identifier of the message to be dequeued ■ <code>condition</code>—dequeue message that satisfy this condition. <p>A condition is specified as a Boolean expression using syntax similar to the <code>WHERE</code> clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the where clause of a SQL query). Message properties include <code>priority</code>, <code>corrid</code> and other columns in the queue table</p> <p>To specify dequeue conditions on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with <code>tab.user_data</code> as a qualifier to indicate the specific column of the queue table that stores the payload. The <code>deq_condition</code> parameter cannot exceed 4000 characters.</p>
	<p>visibility</p> <ul style="list-style-type: none"> ■ <code>ON_COMMIT (default)</code>—The dequeue is part of the current transaction. The operation is complete when the transaction commits. ■ <code>IMMEDIATE</code>—effects of the dequeue are visible immediately after the request is completed. The dequeue is not part of the current transaction. The operation constitutes a transaction on its own.

Table 9–3 IDAP Client Requests for Dequeue—Arguments and Attributes for AQXmlReceive (Cont.)

Argument	Attribute
	<p><code>dequeue_mode</code>—Specifies the locking behavior associated with the dequeue. The <code>dequeue_mode</code> can be specified as one of:</p> <ul style="list-style-type: none"> ■ <code>REMOVE</code> (default): Read the message and update or delete it. This is the default. The message can be retained in the queue table based on the retention properties. ■ <code>BROWSE</code>: Read the message without acquiring any lock on the message. This is equivalent to a select statement. ■ <code>LOCKED</code>: Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This is equivalent to a select for update statement. <p><code>navigation_mode</code>—Specifies the position of the message that will be retrieved. First, the position is determined. Second, the search criterion is applied. Finally, the message is retrieved. The <code>navigation_mode</code> can be specified as one of:</p> <ul style="list-style-type: none"> ■ <code>FIRST_MESSAGE</code>: Retrieves the first message which is available and matches the search criteria. This resets the position to the beginning of the queue. ■ <code>NEXT_MESSAGE</code> (default): Retrieve the next message which is available and matches the search criteria. If the previous message belongs to a message group, then AQ retrieves the next available message which matches the search criteria and belongs to the message group. This is the default. ■ <code>NEXT_TRANSACTION</code>: Skip the remainder of the current transaction group (if any) and retrieve the first message of the next transaction group. This option can only be used if message grouping is enabled for the current queue. <p><code>transformation</code>—the PL/SQL transformation to be invoked after the message is dequeued</p>

The following examples show dequeue requests using different attributes of `AQXmlReceive`.

IDAP Dequeue Request Example 1— Messages from a Single-Consumer Queue

Using the single-consumer queue `QS.NEW_ORDERS_QUE`, the receive request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
```

```

<Body>
  <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
    <consumer_options>
      <destination>QS.NEW_ORDERS_QUE</destination>
      <wait_time>0</wait_time>
    </consumer_options>
  </AQXmlReceive>
</Body>
</Envelope>

```

IDAP Dequeue Request Example 2 — Messages that Satisfy a Specific Condition

Using the multiconsumer queue AQUSER.EMP_TOPIC with subscriber APP1 and condition deptno=60, the receive request has the following format:

```

<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
      <consumer_options>
        <destination>AQUSER.EMP_TOPIC</destination>
        <consumer_name>APP1</consumer_name>
        <wait_time>0</wait_time>
        <selector>
          <condition>tab.user_data.deptno=60</condition>
        </selector>
      </consumer_options>
    </AQXmlReceive>
  </Body>
</Envelope>

```

IDAP Dequeue Request Example 3 — Receiving Messages and Committing

In the dequeue request examples, if you include AQXmlCommit at the end of the RECEIVE request, the transaction is committed upon completion of the operation. In ["IDAP Dequeue Request Example 1— Messages from a Single-Consumer Queue"](#) on page 9-24, the receive request can include the commit flag as follows:

```

<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
      <consumer_options>
        <destination>QS.NEW_ORDERS_QUE</destination>
        <wait_time>0</wait_time>

```

```
        </consumer_options>

        <AQXmlCommit/>

    </AQXmlReceive>
</Body>
</Envelope>
```

IDAP Dequeue Request Example 4 — Browsing Messages

Messages are dequeued in `REMOVE` mode by default. To receive messages from `QS.NEW_ORDERS_QUE` in `BROWSE` mode, modify the receive request as follows:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
    <Body>
        <AQXmlReceive xmlns = "http://ns.oracle.com/AQ/schemas/access">
            <consumer_options>
                <destination>QS.NEW_ORDERS_QUE</destination>
                <wait_time>0</wait_time>
                <dequeue_mode>BROWSE</dequeue_mode>
            </consumer_options>
        </AQXmlReceive>
    </Body>
</Envelope>
```

IDAP Client Requests for Registration

Client requests for registration use the `AQXmlRegister` method. [Table 9-4](#) lists `AQXmlRegister`'s arguments and argument attributes. Required arguments are shown in bold.

Table 9–4 Client Registration—Arguments and Attributes for AQXmlRegister

Argument	Attribute
<code>register_options</code>	<p>destination—specify the queue or topic on which notifications are registered. The destination element has an attribute <code>lookup_type</code> which determines how the destination element value is interpreted</p> <ul style="list-style-type: none"> ■ <code>DATABASE (default)</code>—destination is interpreted as <code>schema.queue_name</code> ■ <code>LDAP</code>—the LDAP server is used to resolve the destination <p>consumer_name—the consumer name for multiconsumer queues or topics. For single consumer queues, this parameter must not be specified</p> <p>notify_url—where notification is sent when a message is enqueued. The form can be <code>http://<url></code> or <code>mailto://<email address></code> or <code>plsql://<pl/sql procedure></code>.</p>

IDAP Register Request Example 1— Registering for Notification at an Email Address

To notify an email address of messages enqueued for consumer APP1 in queue AQUSER.EMP_TOPIC, the register request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns= "http://ns.oracle.com/AQ/schemas/envelope">
  <Body>

    <AQXmlRegister xmlns = "http://ns.oracle.com/AQ/schemas/access">

      <register_options>
        <destination>AQUSER.EMP_TOPIC</destination>
        <consumer_name>APP1</consumer_name>
        <notify_url>mailto:app1@hotmail.com</notify_url>
      </register_options>

      <AQXmlCommit/>

    </AQXmlRegister>
  </Body>
</Envelope>
```

Commit Request

A request to commit all actions performed by the user in a session uses the `AQXmlCommit` method.

Commit Request Example

A commit request has the following format.

```
<?xml version="1.0"?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlCommit xmlns="http://ns.oracle.com/AQ/schemas/access"/>
  </Body>
</Envelope>
```

Rollback Request

A request to roll back all actions performed by the user in a session uses the `AQXmlRollback` method. Actions performed with `IMMEDIATE` visibility are not rolled back.

Rollback Request Example

A rollback request has the following format:

```
<?xml version="1.0"?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlRollback xmlns="http://ns.oracle.com/AQ/schemas/access"/>
  </Body>
</Envelope>
```

IDAP Server Response to Enqueue

The response to an enqueue request to a single-consumer queue uses the `AQXmlSendResponse` method. The components of the response are shown in [Table 9-5](#).

Table 9-5 Server Response to an Enqueue to a Single-Consumer Queue (AQXmlSendResponse)

Response	Attribute
status_response	status_code—indicates success (0) or failure (-1) error_code—Oracle code for the error error_message—description of the error
send_result	destination—where the message was sent message_id—identifier for every message sent

IDAP Server Request Example 1 — Enqueuing to a Single-Consumer Queue

The result of a SEND request to the single consumer queue `QS.NEW_ORDERS_QUE` has the following format:

```
<?xml version = '1.0'?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlSendResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
      <status_response>
        <status_code>0</status_code>
      </status_response>
      <send_result>
        <destination>QS.NEW_ORDERS_QUE</destination>
        <message_id>12341234123412341234</message_id>
      </send_result>
    </AQXmlSendResponse>
  </Body>
</Envelope>
```

The response to an enqueue request to a multiconsumer queue or topic uses the `AQXmlPublishResponse` method. The components of the response are shown in [Table 9-6](#).

Table 9-6 Server Response to an Enqueue to a Multiconsumer Queue or Topic (AQXmlPublishResponse)

Response	Attribute
status_response	status_code—indicates success (0) or failure (-1) error_code—Oracle code for the error error_message—description of the error
publish_result	destination—where the message was sent message_id—identifier for every message sent

IDAP Server Request Example 2— Enqueuing to a Multiconsumer Queue

The result of a SEND request to the multiconsumer queue `AQUSER.EMP_TOPIC` has the following format:

```
<?xml version = '1.0'?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlPublishResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
```

```

<status_response>
  <status_code>0</status_code>
</status_response>
<publish_result>
  <destination>AQUSER.EMP_TOPIC</destination>
  <message_id>23434435435456546546546546</message_id>
</publish_result>
</AQXmlPublishResponse>
</Body>
</Envelope>

```

Server Response to a Dequeue Request

The response to a dequeue request uses the `AQXmlReceiveResponse` method. The components of the response are shown in [Table 9-7](#).

Table 9-7 Server Response to a Dequeue from a Queue or Topic (AQXmlReceiveResponse)

Response	Attribute
status_response	status_code—indicates success (0) or failure (-1) error_code—Oracle code for the error error_message—description of the error
receive_result	destination—where the message was sent message_set—the set of messages dequeued

IDAP Server Dequeue Response Example 1 — Messages from an ADT Queue

The result of a `RECEIVE` request on the queue `AQUSER.EMP_TOPIC` with a payload of type `EMP_TYP` has the following format:

```

<?xml version = '1.0'?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlReceiveResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
      <status_response>
        <status_code>0</status_code>
      </status_response>
      <receive_result>
        <destination>AQUSER.EMP_TOPIC</destination>
        <message_set>
          <message_count>1</message_count>
          <message>

```

```

<message_number>1</message_number>
<message_header>
  <message_id>1234344545565667</message_id>
  <correlation>TKAXAP10</correlation>
  <priority>1</priority>
  <delivery_count>0</delivery_count>
  <sender_id>
    <agent_name>scott</agent_name>
  </sender_id>
  <message_state>0</message_state>
</message_header>
<message_payload>
  <EMP_TYP>
    <EMPNO>1111</EMPNO>
    <ENAME>Mary</ENAME>
    <MGR>5000</MGR>
    <HIREDATE>1996-01-01 0:0:0</HIREDATE>
    <SAL>10000</SAL>
    <COMM>100.12</COMM>
    <DEPTNO>60</DEPTNO>
  </EMP_TYP>
</message_payload>
</message>
</message_set>
</receive_result>
</AQXmlReceiveResponse>
</Body>
</Envelope>

```

Server Response to a Register Request

The response to a register request uses the `AQXmlRegisterResponse` method, which consists of `status_response`. (See [Table 9-7](#) for a description of `status_response`.)

Commit Response

The response to a commit request uses the `AQXmlCommitResponse` method, which consists of `status_response`. (See [Table 9-7](#) for a description of `status_response`.)

Example

The response to a commit request has the following format:

```
<?xml version = '1.0'?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlCommitResponse xmlns="http://ns.oracle.com/AQ/schemas/access">
      <status_response>
        <status_code>0</status_code>
      </status_response>
    </AQXmlCommitResponse>
  </Body>
</Envelope>
```

Rollback Response

The response to a rollback request uses the `AQXmlRollbackResponse` method, which consists of `status_response`. (See [Table 9-7](#) for a description of `status_response`.)

Notification

When an event for which a client has registered occurs, a notification is sent to the client at the URL specified in the `REGISTER` request. `AQXmlNotification` consists of:

- `notification_options`, which has
 - `destination`—the destination queue/topic on which the event occurred
 - `consumer_name`—in case of multiconsumer queues/topics, this refers to the consumer name for which the event occurred
- `message_set`—the set of message properties.

Response in Case of Error

In case of an error in any of the above requests, a `FAULT` is generated. The `FAULT` element consists of:

- `faultcode` - error code for fault
- `faultstring` - indicates a client error or a server error. A client error means that the request is not valid. Server error indicates that the AQ servlet has not been set up correctly

- detail, which consists of
 - `status_response`

IDAP and AQ XML Schemas

IDAP presentation exposes the following two schemas to the client. All documents sent by the Parser are validated against these two schemas:

- **IDAP schema** — `http://ns.oracle.com/AQ/schemas/envelope`
This describes the structure of the document. Each document has an envelope, header, and body.
- **AQ XML schema**. — `http://ns.oracle.com/AQ/schemas/access`
This describes the IDAP body contents for Internet access to AQ features.

See Also: *Oracle9i Application Developer's Guide - Advanced Queuing*

AQXMLServlet

AQXMLServlet is a Java class that extends `oracle.AQ.xml.AQxmlServlet` class. `AQxmlServlet` class in turn extends `javax.servlet.http.HttpServlet` class.

See Also: *Oracle9i Application Developer's Guide - Advanced Queuing* for information on creating and deploying AQ XML Servlet.

Accessing AQXMLServlet with HTTP

See: *Oracle9i Application Developer's Guide - Advanced Queuing*, for setting up AQ to receive XML messages over HTTP.

How AQ Client Makes a Request to AQ Servlet Using HTTP

The general AQ client procedure making a request using HTTP to the AQ Servlet, is as follows:

1. The AQ client opens an HTTP(S) connection to the server. For example:
`https://aq.us.oracle.com:8000/aqserv/servlet/AQTestServlet`
This opens a connection to port 8000 on `aq.us.oracle.com`
2. The AQ client logs in to the server by either:

- HTTP basic authentication (with or without SSL)
 - SSL certificate based client authentication.
3. The AQ client constructs the XML message representing the Send, Publish, Receive or Register request. For example:

```
<?xml version="1.0"?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlSend xmlns="http://ns.oracle.com/AQ/schemas/access">
      <producer_options>
        <destination>OE.OE_NEW_ORDERS_QUE</destination>
      </producer_options>
      <message_set>
        <message_count>1</message_count>
        <message>
          <message_number>1</message_number>
          <message_header>
            <correlation>XML_ADT_SINGLE_ENQ</correlation>
            <sender_id>
              <agent_name>john</agent_name>
            </sender_id>
          </message_header>
          <message_payload>
            <ORDER_TYP>
              <ORDERNO>100</ORDERNO>
              <STATUS>NEW</STATUS>
              <ORDERTYPE>NORMAL</ORDERTYPE>
              <ORDERREGION>EAST</ORDERREGION>
              <CUSTOMER>
                <CUSINO>1001233</CUSINO>
                <CUSTID>JOHN</CUSTID>
                <NAME>AMERICAN EXPRESS</NAME>
                <STREET>EXPRESS STREET</STREET>
                <CITY>REDWOOD CITY</CITY>
                <STATE>CA</STATE>
                <ZIP>94065</ZIP>
                <COUNTRY>USA</COUNTRY>
              </CUSTOMER>
              <PAYMENTMETHOD>CREDIT</PAYMENTMETHOD>
            </ORDER_TYP>
            <ITEMS>
              <ITEMS_ITEM>
                <QUANTITY>10</QUANTITY>
                <ITEM>
                  <TITLE>Perl</TITLE>
                </ITEM>
              </ITEMS_ITEM>
            </ITEMS>
          </message_payload>
        </message>
      </message_set>
    </AQXmlSend>
  </Body>
</Envelope>
```



```

                <AUTHORS>Randal</AUTHORS>
                <ISBN>ISBN20200</ISBN>
                <PRICE>19</PRICE>
            </ITEM>
            <SUBTOTAL>190</SUBTOTAL>
        </ITEMS_ITEM>
    </ITEMS>
    <CCNUMBER>NUMBER01</CCNUMBER>
    <ORDER_DATE>2000-08-23 0:0:0</ORDER_DATE>
</ORDER_TYP>
</message_payload>
</message>
</message_set>
</AQxmlSend>
</Body>
</Envelope>

```

4. The AQ client sends an HTTP POST to the Servlet at the remote server.

How AQ Servlet Processes a Request Using HTTP

The AQ servlet's general procedure for making a request using HTTP is as follows:

1. The server accepts the client HTTP(S) connection
2. The server authenticates the user (AQ agent) specified by the client
3. The server receives the POST request
4. AQ servlet is invoked. If this is the first request being serviced by this servlet, the servlet is initialized - its `init()` method is invoked. The `init()` method creates a connection pool to the Oracle server using the `AQxmlDataSource` parameters (sid, host, port, aq servlet super-user name, password) provided by the client.
5. AQ servlet processes the message as follows:
 - a. If this is the first request from this client, a new HTTP session is created. The XML message is parsed and its contents are validated. If a SessionID is passed by the client in the HTTP headers, then this operation is performed in the context of that session - this is described in detail in *Oracle9i Application Developer's Guide - Advanced Queuing*
 - b. The servlet determines which object (queue/topic) the agent is trying to perform operations on. For example, in the above request sequence (Step 3

in "How AQ Client Makes a Request to AQ Servlet Using HTTP"), the agent "JOHN" is trying to access the OE.OE_NEW_ORDERS_QUE.

- c. After that the servlet looks through the list of database users that map to this AQ Agent (using the AQ\$INTERNET_USERS view). If any one of these db_users has privileges to access the queue/topic specified in the request, the aq servlet super-user creates a session on behalf of this db_user.
- d. For example, in the above example, say, "JOHN" was mapped to the database user "OE" using the DBMS_AQADM.ENABLE_DB_ACCESS call. The servlet will create a session for the agent "JOHN" with the privileges of database user OE.
- e. If there is no transaction active in this HTTP session, then a new database transaction is started. Subsequent requests in this session will be part of the same transaction until an explicit commit or rollback request is made.
- f. Now the requested operation (send/publish/receive/register) is performed.
- g. The response is formatted as an XML message and sent back the client. For exampl, the response for the above request could be:

```
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
  <Body>
    <AQXmlSendResponse
      xmlns="http://ns.oracle.com/AQ/schemas/access">
      <status_response>
        <status_code>0</status_code>
      </status_response>
      <send_result>
        <destination>OE.OE_NEW_ORDERS_QUE</destination>
        <message_id>12341234123412341234123412341234</message_id>
      </send_result>
    </AQXmlSendResponse>
  </Body>
</Envelope>
```

The response also includes the session id in the HTTP headers as a cookie. For example: Tomcat sends back session ids as JSESSIONID=239454ds2343

XMLType Queues

Storing and Querying XML Documents with Advanced Queueing (AQ)

Advanced Queueing (AQ) supports the storage of XML documents in queues and provides the ability to query the XML documents. XML can be used with Oracle AQ in the following two cases:

- **XML data stored in queues.** XML payloads are supported by AQ queues. XML messages can be stored as the XMLType datatype.
- **XML data generated from existing queues with ADT or RAW payloads.** Used by applications that already store their data in ADT or RAW queues and where new applications, written on the same data, need to use XML as the message format.

See Also: [Chapter 5, "Database Support for XML"](#), for more details on XML support in the database.

Structuring and Managing Message Payloads with Object Types

With Oracle AQ, you can use object types to structure and manage the payload of messages. Using strongly typed content, content whose format is defined by an external type system, the following AQ features are made available:

- **Content-based routing:** AQ can examine the content and automatically route messages to another queue based on content.
- **Content-based subscription:** A publish and subscribe system can be built on top of a messaging system so that you can create subscriptions based on content.
- **Querying:** The ability to execute queries on the content of messages allows you to examine current and processed messages for various applications, including message warehousing.

Creating Message Payloads Queues Containing XMLType Attributes

You can create queues with payloads that contain XMLType attributes. These can be used for transmitting and storing messages that contain XML documents. By defining Oracle objects with XMLType attributes, you can do the following:

- Store more than one type of XML document in the same queue. The documents are stored internally as CLOBs.

- Selectively dequeue messages with XMLType attributes using the operators `XMLType.existsNode()`, `XMLType.extract()`, and so on.

See Also: *Oracle9i Application Developer's Guide - XML* for details on XMLType operations.

- Define transformations to convert Oracle objects to XMLType.
- Define rule-based subscribers that query message content using XMLType operators such as `XMLType.existsNode()` and `XMLType.extract()`.

XMLType Queues Example 1: Creating XMLType Queue Tables for a Queue Object

In the BooksOnline application, assume that the Overseas Shipping site represents the order as `ORDER_XML_TYP`, with the order information in an XMLType attribute. The Order Entry site represents the order as an Oracle object, `ORDER_TYP`.

`ORDER_XML_TYP` is a composite type that contains an XMLType attribute:

```
CREATE OR REPLACE TYPE order_xml_typ as OBJECT (  
    orderno    NUMBER,  
    details    XMLTYPE);
```

The Overseas queue table and queue are created as follows:

```
BEGIN  
dbms_aqadm.create_queue_table(  
    queue_table      => 'OS_orders_pr_mqtab',  
    comment          => 'Overseas Shipping MultiConsumer Orders queue table',  
    multiple_consumers => TRUE,  
    queue_payload_type => 'OS.order_xml_typ',  
    compatible       => '8.1');  
END;  
  
BEGIN  
dbms_aqadm.create_queue (  
    queue_name       => 'OS_bookedorders_que',  
    queue_table      => 'OS_orders_pr_mqtab');  
END;
```

AQ XML Message Format Transformation

You can specify transformations between different Oracle and user-defined types. Transformations can be created in any of the following ways:

- PL/SQL functions (including callouts). See [AQ Message Transformation Example 1: Creating a PL/SQL Function](#)
- SQL expressions
- Java stored procedures

with a return type of the target type.

Only one-to-one message transformation is supported. The transformation engine is integrated with Advanced Queuing to facilitate transformation of messages as they move through the database messaging system.

An Advanced Queuing application can enqueue or dequeue messages from a queue in the format specified by the application. An application can also specify a message format when subscribing to queues.

The AQ propagator transforms messages to the format of the destination queue message, as specified by the remote subscription. The transformation function cannot write the database state or commit/rollback the current transaction. Transformations are exported with a schema or a full database export.

AQ Message Transformation Example 1: Creating a PL/SQL Function

An Order Entry site represents the order as Oracle object, `ORDER_TYP`.

Since the Overseas Shipping site subscribes to messages in the `OE_BOOKEDORDERS_QUE` queue, a transformation is applied before messages are propagated from the Order Entry site to the Overseas Shipping site.

`ORDER_XML_TYP` is a composite type that contains an `XMLType` attribute:

```
CREATE OR REPLACE TYPE order_xml_typ as OBJECT (
    orderno NUMBER,
    details XMLTYPE);
```

The transformation is defined as follows:

```
CREATE OR REPLACE FUNCTION CONVERT_TO_ORDER_XML(input_order TYPE OE.ORDER_TYP)
RETURN OS.ORDER_XML_TYP AS
    xdata SYS.XMLType;
    new_order OS.ORDER_XML_TYP;
BEGIN
    xdata := XMLType.createXML(input_order, NULL);
    new_order := OS.ORDER_XML_TYP(input_order.orderno, xdata);
    RETURN new_order;
END CONVERT_TO_ORDER_XML;
```

```
execute dbms_transform.create_transformation(
    schema =>      'OS',
    name   =>      'OE2XML',
    from_schema => 'OE',
    from_type =>   'ORDER_TYP',
    to_schema =>   'OS',
    to_type  =>   'ORDER_XML_TYP',
    transformation => 'CONVERT_TO_ORDER_XML(source.user_data)');

/* Add a rule-based subscriber for Overseas Shipping to the Booked orders
queues with Transformation. Overseas Shipping handles all non-US orders: */
DECLARE
    subscriber    aq$_agent;
BEGIN
    subscriber := aq$_agent('Overseas_Shipping', 'OS.OS_bookedorders_que', null);

    dbms_aqadm.add_subscriber(
        queue_name    => 'OE.OE_bookedorders_que',
        subscriber    => subscriber,
        rule          => 'tab.user_data.orderregion = ''INTERNATIONAL''',
        transformation => 'OS.OE2XML');
END;
```

Assume that an application processes orders for customers in Canada. This application can dequeue messages using the following procedure:

```
/* Create procedures to enqueue into single-consumer queues: */
create or replace procedure get_canada_orders() as
deq_msgid          RAW(16);
dopt               dbms_aq.dequeue_options_t;
mprop              dbms_aq.message_properties_t;
deq_order_data    OS.order_xml_typ;
no_messages        exception;
pragma exception_init (no_messages, -25228);
new_orders         BOOLEAN := TRUE;

begin
    dopt.wait := 1;

/* Specify dequeue condition to select Orders for Canada */
    dopt.deq_condition := 'tab.user_data.xdata.extract(
''/ORDER_TYP/CUSTOMER/COUNTRY/text()'').getStringVal()='CANADA''';

    dopt.consumer_name := 'Overseas_Shipping';
```

```
WHILE (new_orders) LOOP
  BEGIN
    dbms_aq.dequeue(
      queue_name          => 'OS.OS_bookedorders_que',
      dequeue_options     => dopt,
      message_properties  => mprop,
      payload             => deq_order_data,
      msgid               => deq_msgid);
    commit;

    dbms_output.put_line(' Order for Canada - Order No: ' ||
      deq_order_data.orderno);

  EXCEPTION
    WHEN no_messages THEN
      dbms_output.put_line (' ---- NO MORE ORDERS ---- ');
      new_orders := FALSE;
  END;
END LOOP;

end;
```

See Also

- *Oracle9i Application Developer's Guide - Advanced Queuing*, Chapter 8, for more detail on how to implement structured message payloads applications using either DBMS_AQADM or Java (JDBC)
- *Oracle9i Supplied PL/SQL Packages Reference* for more information about DBMS_TRANSFORM.

Frequently Asked Questions (FAQs): XML and Advanced Queuing

Can we Store AQ XML Messages with Many PDFs as One Record?

Question

We are exchanging XML documents from one business area to another using Oracle Advanced Queuing. Each message received or sent includes an XML header, XML attachment (XML data stream), DTDs, and PDF files. We need to store all this

information, including some imagery files, in the database table, in this case, the queuetable.

Can we enqueue this message into an Oracle queue table as one record or one piece? Or do we have to enqueue this message as multiple records, such as one record for XML data streams as CLOB type, one record for PDF files as RAW type? Then somehow specify that these sets of records are correlated? Also, we want to ensure that we dequeue this.

Answer

You can achieve this in the following ways:

- You can either define an object type with (CLOB, RAW,...) attributes, and store it as a single message
- You can use the AQ message grouping feature and store it in multiple messages. But the message properties will be associated with a group. To use the message grouping feature, all messages must be the same payload type.

Question 2

Does this mean that we specify the payload type as CLOB first, then enqueue and store all the pieces, XML message data stream, DTDs, and PDF,... as a single message payload in the Queue table? If so, how can we separate this single message into individual pieces when we dequeue this message?

Answer 2

No. You create an object type, for example:

```
CREATE TYPE mypayload_type as OBJECT (xmlDataStream CLOB, dtd CLOB, pdf BLOB);
```

Then store it as a single message.

Can We Add New Recipients After Messages are Enqueued?

Question

We want to use the queue table to support message assignments. For example, when other business areas send messages to Oracle, they do not know who should be assigned to process these messages, but they know the messages are for Human Resources (HR). So all messages will go to the HR supervisor.

At this point, the message has been enqueued in the queue table. The HR supervisor is the only recipient of this message, and the entire HR staff have been pre-defined as subscribers for this queue). Can the HR supervisor add new recipients, namely additional staff, to the message_properties.recipient_list on the existing the message in the queue table?

We do not have multiple consumers (recipients) when the messages are enqueued, but we want to replace the old recipient, or add new recipients after the message has already been in the queue table. This new message will then be dequeued by the new recipient. Is this workable? Or do we have to remove the message from old recipient, then enqueue the same message contents to the new recipient?

Answer

You cannot change the recipient list after the message is enqueued. If you do not specify a recipient list then subscribers can subscribe to the queue and dequeue the message.

In your case, the new recipient should be a subscriber to the queue. Otherwise, you will have to dequeue the message and enqueue it again with the new recipient.

How Does Oracle Enqueue and Dequeue and Process XML Messages?

Question

In the OTN document, “Using XML in Oracle Database Applications, Part 4, Exchanging Business Data Among Applications” Nov. 1999, it says that an Oracle database can enqueue and dequeue XML messages and process them. How does it do this?

Do I have to use XML SQL Utility (XSU) in order to insert an XML file into a table before processing it, or can I enqueue an XML file directly, parse it, and dispatch its messages via the AQ process? Must I use XML SQL Utility every time I want to INSERT or UPDATE XML data into an Oracle Database?

Answer

AQ supports enqueueing and dequeuing objects. These objects can have an attribute of type XMLType containing an XML Document, as well as other interested “factored out” metadata attributes that might make sense to send along with the message. Refer to the latest AQ document, *Oracle8i Application Developer’s Guide - Advanced Queuing*, to get specific details and see more examples.

How Can We Parse Messages with XML Content From AQ Queues?

Question

We need a tool to parse messages with XML content, from an AQ queue and then update tables and fields in an ODS (Operational Data Store). In short, we want to retrieve and parse XML documents and map specific fields to database tables and columns.

Is Oracle Text (*intermedia* Text/Context) a solution?

Answer

The easiest way to do this is using Oracle XML Parser for Java and Java Stored Procedures in tandem with AQ inside Oracle.

Question 2

We can use XML SQL Utility if we go with a custom solution. Our main concentration is supply-chain. We want to get metadata information such as, AQ enqueue/dequeue times, JMS header information,... based on queries on certain XML tag values. Can we just store the XML in a CLOB and issue queries using Oracle Text (*intermedia* Text)?

Answer 2

- If you store XML as CLOBs then you can definitely search it using Oracle Text (*interMedia* Text), but this only helps you find a particular message that matches a criteria.
- If you need to do aggregation operations over the metadata, view the metadata from existing relational tools, or use normal SQL predicates on the metadata, then having it “only” stored as XML in a CLOB is not going to be good enough.

You can combine Oracle Text (*interMedia* Text) XML searching with some amount of redundant metadata storage as “factored out” columns and use SQL statements that combine normal SQL predicates with the Oracle Text (*interMedia* Text) CONTAINS() clause to have the best of both.

See Also: [Chapter 8, "Searching XML Data with Oracle Text"](#).

Can we Prevent the Listener From Stopping Until the XML Document is Processed?

Question

We receive XML messages from clients as messages and need to process them as soon as they come in. Each XML document takes about 15 seconds to process. We are using PL/SQL.

One PL/SQL procedure starts the listener and Dequeues the message and calls another procedure to process the XML document. The problem is that the listener is held up until the XML document is processed. Meanwhile messages accumulate in the queue.

What is the best way to handle this? Is there a way for the listener program to call the XML processing procedure asynchronously and return to listening? Java is not an option at this point.

Answer

After receiving the message, you can submit a job using the DBMS_JOB package. The job will be invoked asynchronously in a different database session.

Oracle has added PL/SQL callbacks in the AQ notification framework. This allows you register a PL/SQL callback which is invoked asynchronously when a message shows up in a queue.

Part IV

Tools and Frameworks for Building Oracle-Based XML Applications

This section includes a description of how to use XSQL Servlet Pages. XSQL Servlet is part of XDK for Java.

Other chapters in Part IV describe how to use JDeveloper, BC4J, Metadata API, Oracle Reports, and Oracle Portal, to build Oracle-based XML applications. It also introduces you to Oracle Exchange and Oracle XML Gateway.

Part IV contains the following chapters:

- [Chapter 10, "XSQL Pages Publishing Framework"](#)
- [Chapter 11, "Using JDeveloper to Build Oracle XML Applications"](#)
- [Chapter 12, "Building BC4J and XML Applications"](#)
- [Chapter 13, "Using Metadata API"](#)
- [Chapter 14, "OracleAS Reports Services and XML"](#)
- [Chapter 15, "Using the PDK for Visualizing XML Data in Oracle Portal"](#)
- [Chapter 16, "How Oracle Exchange Uses XML"](#)
- [Chapter 17, "Introducing Oracle XML Gateway"](#)

XSQL Pages Publishing Framework

This chapter contains the following sections:

- XSQL Pages Publishing Framework Overview
 - What Can I Do with Oracle XSQL Pages?
 - Where Can I Obtain Oracle XSQL Pages?
 - What's Needed to Run XSQL Pages?
- Overview of Basic XSQL Pages Features
 - Producing XML Datagrams from SQL Queries
 - Transforming XML Datagrams into an Alternative XML Format
 - Transforming XML Datagrams into HTML for Display
- Setting Up and Using XSQL Pages in Your Environment
 - Using XSQL Pages With Oracle JDeveloper
 - Setting the CLASSPATH Correctly in Your Production Environment
 - Setting Up the Connection Definitions
 - Using the XSQL Command Line Utility
- Overview of All XSQL Pages Capabilities
 - Using All of the Core Built-in Actions
 - Aggregating Information Using `<xsql:include-xsql>`
 - Handling Posted Information
 - Using Custom XSQL Action Handlers
- Description of XSQL Servlet Examples

- [Setting Up the Demo Data](#)
- [Advanced XSQL Pages Topics](#)
 - [Understanding Client Stylesheet-Override Options](#)
 - [Controlling How Stylesheets are Processed](#)
 - [Using XSQLConfig.xml to Tune Your Environment](#)
 - [Using the FOP Serializer to Produce PDF Output](#)
 - [Using XSQL Page Processor Programmatically](#)
 - [Writing Custom XSQL Action Handlers](#)
 - [Writing Custom XSQL Serializers](#)
 - [Writing Custom XSQL Connection Managers](#)
 - [Formatting XSQL Action Handler Errors](#)
- [XSQL Servlet Limitations](#)
- [Frequently Asked Questions \(FAQs\) - XSQL Servlet](#)

XSQL Pages Publishing Framework Overview

The Oracle XSQL Pages publishing framework is an extensible platform for easily publishing XML information in any format you desire. It greatly simplifies combining the power of SQL, XML, and XSLT to publish dynamic web content based on database information.

Using the XSQL publishing framework, anyone familiar with SQL can create and use declarative templates called "XSQL pages" to:

- Assemble dynamic XML "datagrams" based on parameterized SQL queries, and
- Transform these "data pages" to produce a final result in any desired XML, HTML, or text-based format using an associated XSLT transformation.

Assembling and transforming information for publishing requires no programming. In fact, most of the common things you will want to do can be easily achieved in a declarative way. However, since the XSQL publishing framework is extensible, if one of the built-in features does not fit your needs, you can easily extend the framework using Java to integrate custom information sources or to perform custom server-side processing.

Using the XSQL Pages framework, the *assembly* of information to be published is cleanly separated from presentation. This simple architectural detail has profound productivity benefits. It allows you to:

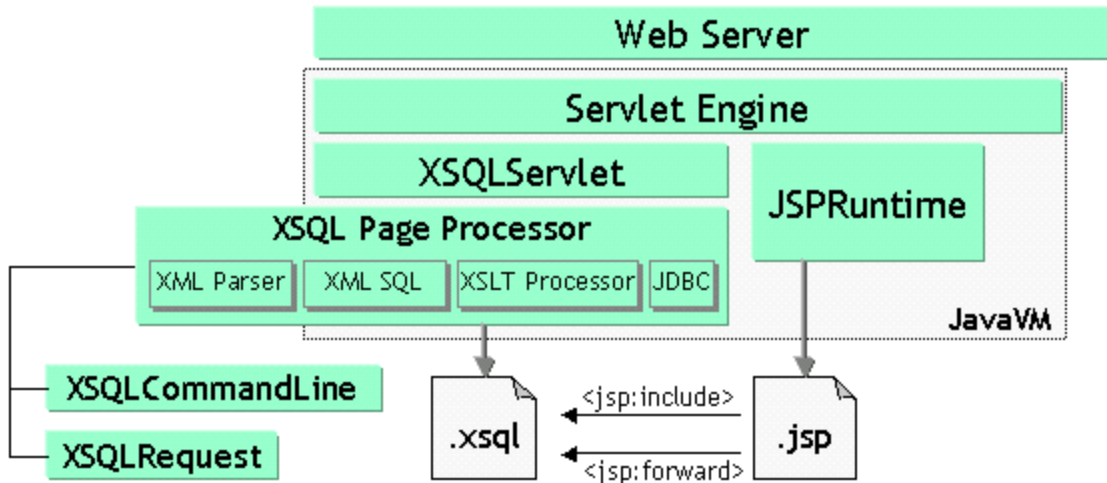
- Present the same information in multiple ways, including tailoring the presentation appropriately to the kind of client device making the request (browser, cellular phone, PDA, etc.).
- Reuse information easily by aggregating existing pages into new ones
- Revise and enhance the presentation independently of the information content being presented.

What Can I Do with Oracle XSQL Pages?

Using server-side templates — known as "XSQL pages" due to their `.xsql` extension — you can publish any information in any format to any device. The XSQL page processor "engine" interprets, caches, and processes the contents of your XSQL page templates. [Figure 10–1](#) illustrates that the core XSQL page processor engine can be "exercised" in four different ways:

- From the command line or in batch using the *XSQL Command Line Utility*
- Over the Web, using the *XSQL Servlet* installed into your favorite web server
- As part of JSP applications, using `<jsp:include>` to include a template
- Programmatically, with the `XSQLRequest` object, the engine's Java API

Figure 10–1 Understanding the Architecture of the XSQL Pages Framework



The same XSQL page templates can be used in any or all of these scenarios. Regardless of the means by which a template is processed, the same basic steps occur to produce a result. The XSQL page processor "engine":

1. Receives a request to process an XSQL template
2. Assembles an XML "datagram" using the result of one or more SQL queries
3. Returns this XML "datagram" to the requestor
4. Optionally transforms the "datagram" into any XML, HTML, or text format

During the transformation step in this process, you can use stylesheets that conform to the W3C XSLT 1.0 standard to transform the assembled "datagram" into document formats like:

- HTML for browser display
- Wireless Markup Language (WML) for wireless devices
- Scalable Vector Graphics (SVG) for data-driven charts, graphs, and diagrams
- XML Stylesheet Formatting Objects (XSL-FO), for rendering into Adobe PDF
- Text documents, like emails, SQL scripts, Java programs, etc.
- Arbitrary XML-based document formats

XSQL Pages bring this functionality to you by automating the use of underlying Oracle XML components to solve many common cases without resorting to custom programming. However, when only custom programming will do — as we'll see in the Advanced Topics section of this chapter — you can augment the framework's built-in actions and serializers to assemble the XSQL "datagrams" from any custom source and serialize the datagrams into any desired format, without having to write an entire publishing framework from scratch.

See Also:

- [Appendix C, "XDK for Java: Specifications and Cheat Sheets"](#) for the XSQL Servlet specifications and cheat sheets
- XSQL Servlet Release Notes on OTN at: <http://otn.oracle.com/tech/xml>

Where Can I Obtain Oracle XSQL Pages?

XSQL Servlet is provided with Oracle and is also available for download from the OTN site: <http://otn.oracle.com/tech/xml>.

Where indicated, the examples and demos described in this chapter are also available from OTN.

What's Needed to Run XSQL Pages?

To run the Oracle XSQL Pages publishing framework from the command-line, all you need is a Java VM (1.1.8, 1.2.2, or 1.3). The XSQL Pages framework depends on and comes bundled with two underlying components in the Oracle XML Developer's Kit:

- Oracle XML Parser and XSLT Processor (`xmlparserv2.jar`)
- Oracle XML SQL Utility (`xsu12.jar`)

Both of their Java archive files must be present in the CLASSPATH where the XSQL pages framework is running. Since most XSQL pages will connect to a database to query information for publishing, the framework also depends on a JDBC driver. Any JDBC driver is supported, but when connecting to Oracle, it's best to use the Oracle JDBC driver (`classes12.zip`) for maximum functionality and performance.

Lastly, the XSQL publishing engine expects to read its configuration file named `XSQLConfig.xml` as a Java resource, so you must include the *directory* where the `XSQLConfig.xml` file resides in the CLASSPATH as well.

To use the XSQL Pages framework for Web publishing, in addition to the above you'll need a web server that supports Java Servlets. The following is the list of web servers with Servlet capability on which the XSQL Servlet has been tested:

- Oracle9i Internet Application Server v1.x and v2.x
- Oracle9i Oracle Servlet Engine
- Allaire JRun 2.3.3 and 3.0.0
- Apache 1.3.9 or higher with JServ 1.0/1.1 or Tomcat 3.1/3.2 Servlet Engine
- Apache Tomcat 3.1 or 3.2 Web Server + Servlet Engine
- Caucho Resin 1.1
- Java Web Server 2.0
- Weblogic 5.1 Web Server
- NewAtlanta ServletExec 2.2 and 3.0 for IIS/PWS 4.0
- Oracle8i Lite Web-to-Go Server
- Sun JavaServer Web Development Kit (JSWDK) 1.0.1 Web Server

Note: For security reasons, when installing XSQL Servlet on your production web server, make sure `XSQLConfig.xml` file does *not* reside in a directory that is part of the web server's virtual directory hierarchy. Failure to take this precaution risks exposing your configuration information over the web.

For details on installing, configuring your environment, and running XSQL Servlet and for additional examples and guidelines, see the XSQL Servlet "Release Notes" on OTN at <http://otn.oracle.com/tech/xml>

Overview of Basic XSQL Pages Features

In this section, we'll get take a brief look at the most basic features you can exploit in your server-side XSQL page templates:

- Producing XML Datagrams from SQL Queries

- Transforming the XML Datagram into an Alternative XML Format
- Transforming the XML Datagram into HTML for Display

Producing XML Datagrams from SQL Queries

It is extremely easy to serve database information in XML format over the Web using XSQL pages. For example, let's see how simple it is to serve a real-time XML "datagram" from Oracle, of all available flights landing today at JFK airport. Using Oracle JDeveloper — or your favorite text editor — just build an XSQL page template like the one below, and save it in a file named, `AvailableFlightsToday.xsql`:

```
<?xml version="1.0"?>
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
    SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime, 'HH24:MI') AS Due
    FROM FlightSchedule
    WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
    AND Destination = ? /* The ? is a bind variable being bound */
    ORDER BY ExpectedTime /* to the value of the City parameter */
</xsql:query>
```

With XSQL Servlet properly installed on your web server, you just need to copy the `AvailableFlightsToday.xsql` file above to a directory under your web server's virtual directory hierarchy. Then you can access the template through a web browser by requesting the URL:

```
http://yourcompany.com/AvailableFlightsToday.xsql?City=JFK
```

The results of the query in your XSQL page are materialized automatically as XML and returned to the requestor. This XML-based "datagram" would typically be requested by another server program for processing, but if you are using a browser such as Internet Explorer 5.0, you can directly view the XML result as shown in [Figure 10-2](#).

Figure 10–2 XML Result From XSQL Page (AvailableFlightsToday.xsql) Query

```

<?xml version="1.0" ?>
- <ROWSET>
  - <ROW num="1">
    <CARRIER>VS</CARRIER>
    <FLIGHTNUMBER>344</FLIGHTNUMBER>
    <ORIGIN>London</ORIGIN>
    <DUE>16:10</DUE>
  </ROW>
  - <ROW num="2">
    <CARRIER>LH</CARRIER>
    <FLIGHTNUMBER>466</FLIGHTNUMBER>
    <ORIGIN>Frankfurt</ORIGIN>
    <DUE>21:33</DUE>
  </ROW>
  - <ROW num="3">
    <CARRIER>UA</CARRIER>
    <FLIGHTNUMBER>32</FLIGHTNUMBER>
    <ORIGIN>San Francisco</ORIGIN>
    <DUE>23:54</DUE>
  </ROW>
</ROWSET>

```

Let's take a closer look at the "anatomy" of the XSQL page template we used. Notice the XSQL page begins with:

```
<?xml version="1.0"?>
```

This is because the XSQL template is itself an XML file (with an *.xsql extension) that contains any mix of static XML content and XSQL "action elements". The AvailableFlightsToday.xsql example above contains no *static* XML elements, and just a single XSQL action element `<xsql:query>`. It represents the simplest useful XSQL page we can build, one that just contains a single query.

Notice that the first (and in this case, only!) element in the page `<xsql:query>` includes a special attribute that declares the `xsql` namespace prefix as a "nickname" for the Oracle XSQL namespace identifier `urn:oracle-xsql`.

```
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
```

This first, outermost element — known at the "document element" — also contains a `connection` attribute whose value "demo" is the name of one of the pre-defined connections in the `XSQLConfig.xml` configuration file:

```
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
```

The details concerning the username, password, database, and JDBC driver that will be used for the "demo" connection are centralized into the configuration file. Setting up these connection definitions is discussed in a later section of this chapter.

Lastly, the `<xsql:query>` element contains a `bind-params` attribute that associates the values of parameters in the request by name to bind parameters represented by question marks in the SQL statement contained inside the `<xsql:query>` tag.

Note that if we wanted to include more than one query on the page, we'll need to invent an XML element of our own creation to "wrap" the other elements like this:

```
<?xml version="1.0"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query bind-params="City">
    SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') AS Due
    FROM FlightSchedule
    WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
    AND Destination = ? /* The ? is a bind variable being bound */
    ORDER BY ExpectedTime /* to the value of the City parameter */
  </xsql:query>
  <!-- Other xsql:query actions can go here inside <page> and </page> -->
</page>
```

Notice in this example that the `connection` attribute and the `xsql` namespace declaration *always* go on the document element, while the `bind-params` is specific to the `<xsql:query>` action.

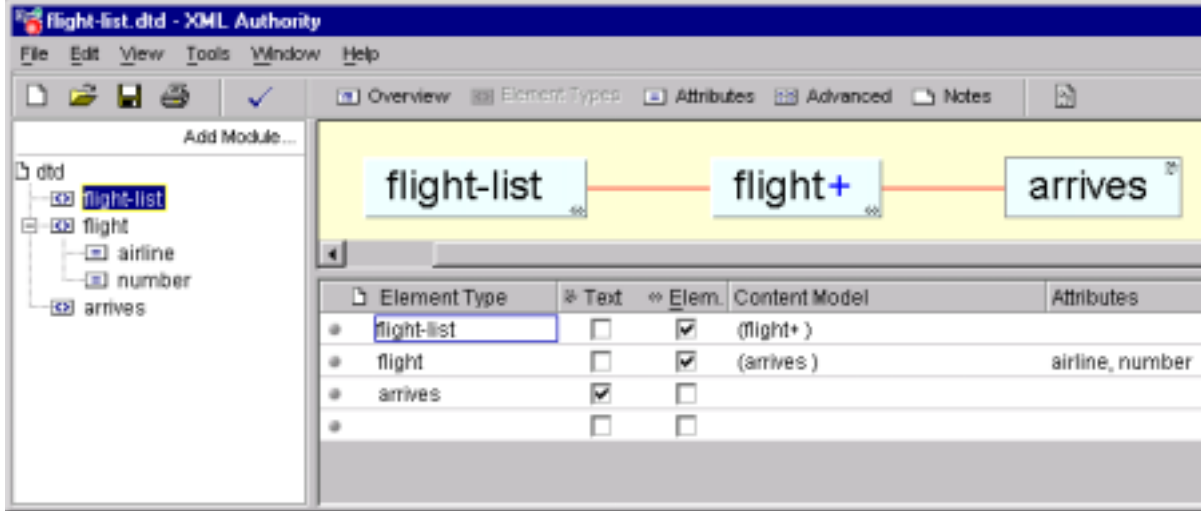
Transforming XML Datagrams into an Alternative XML Format

If the canonical `<ROWSET>` and `<ROW>` XML output from [Figure 10-2](#) is not the XML format you need, then you can associate an XSLT stylesheet to your XSQL page template to transform this XML "datagram" in the server before returning the information in any alternative format desired.

When exchanging data with another program, typically you will agree in advance with the other party on a specific Document Type Descriptor (DTD) that describes the XML format you will be exchanging. A DTD is in effect, a "schema" definition. It formally defines what XML elements and attributes that a document of that type can have.

Let's assume you are given the `flight-list.dtd` definition and are told to produce your list of arriving flights in a format compliant with that DTD. You can use a visual tool such as Extensibility's "XML Authority" to browse the structure of the `flight-list` DTD as shown in [Figure 10-3](#).

Figure 10-3 Exploring the 'industry standard' `flight-list.dtd` using Extensibility's XML Authority



This shows that the standard XML formats for Flight Lists are:

- `<flight-list>` element, containing one or more...
- `<flight>` elements, having attributes `airline` and `number`, each of which contains an...
- `<arrives>` element.

By associating the following XSLT stylesheet, `flight-list.xsl`, with the XSQL page, you can "morph" the default `<ROWSET>` and `<ROW>` format of your arriving flights into the "industry standard" DTD format.


```

<!-- XSLT Stylesheet to transform ROWSET/ROW results into flight-list format -->
<flight-list xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            xsl:version="1.0">
  <xsl:for-each select="ROWSET/ROW">
    <flight airline="{CARRIER}" number="{FLIGHTNUMBER}">
      <arrives><xsl:value-of select="DUE"/></arrives>
    </flight>
  </xsl:for-each>
</flight-list>

```

The stylesheet is a template that includes the literal elements that you want produced in the resulting document, such as, <flight-list>, <flight>, and <arrives>, interspersed with special XSLT "actions" that allow you to do the following:

- Loop over matching elements in the source document using <xsl:for-each>
- Plug in the values of source document elements where necessary using <xsl:value-of>
- Plug in the values of source document elements into attribute values using {something}

Note two things have been added to the top-level <flight-list> element in the stylesheet:

- `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
This defines the XML Namespace (xmlns) named "xsl" and identifies the uniform resource locator string that uniquely identifies the XSLT specification. Although it looks just like a URL, think of the string `http://www.w3.org/1999/XSL/Transform` as the "global primary key" for the set of elements that are defined in the XSLT 1.0 specification. Once the namespace is defined, we can then make use of the <xsl:XXX> action elements in our stylesheet to loop and plug values in where necessary.
- `xsl:version="1.0"`
This attribute identifies the document as an XSLT 1.0 stylesheet. A version attribute is required on all XSLT Stylesheets for them to be valid and recognized by an XSLT Processor.

Associate the stylesheet to your XSQL Page by adding an <?xml-stylesheet?> processing instruction to the top of the page as follows:

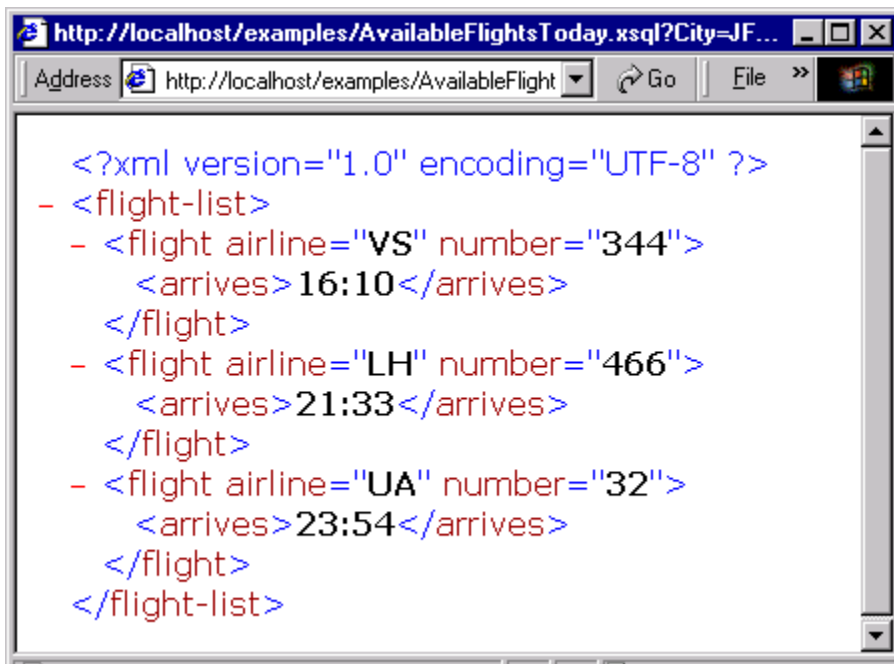
```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="flight-list.xsl"?>
<xsql:query connection="demo" bind-params="City" xmlns:xsql="urn:oracle-xsql">
  SELECT Carrier, FlightNumber, Origin, TO_CHAR(ExpectedTime,'HH24:MI') AS Due
  FROM FlightSchedule
  WHERE TRUNC(ExpectedTime) = TRUNC(SYSDATE) AND Arrived = 'N'
  AND Destination = ? /* The ? is a bind variable being bound */
  ORDER BY ExpectedTime /* to the value of the City parameter */
</xsql:query>

```

This is the W3C Standard mechanism of associating stylesheets with XML documents (<http://www.w3.org/TR/xml-stylesheet>). Specifying an associated XSLT stylesheet to the XSQL page causes the requesting program or browser to see the XML in the “industry-standard” format as specified by `flight-list.dtd` you were given as shown in [Figure 10-4](#).

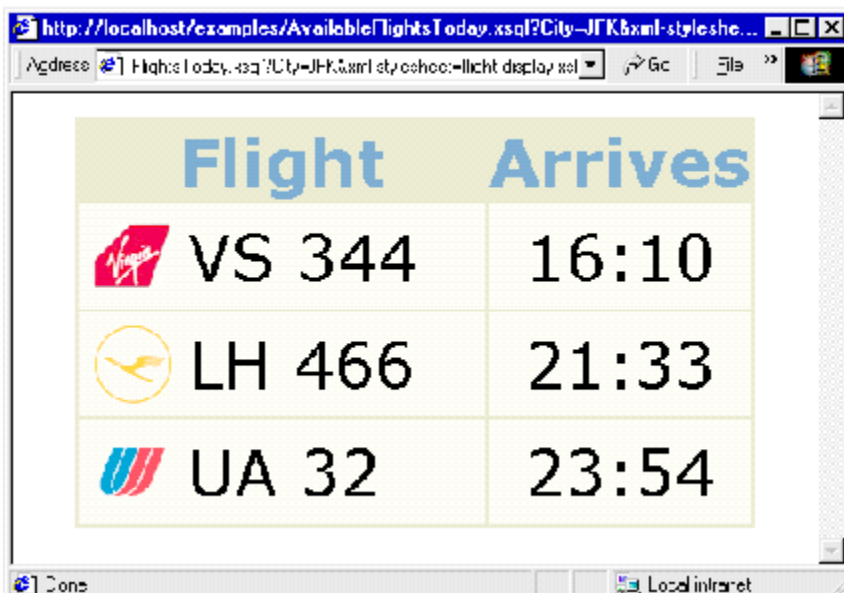
Figure 10-4 XSQL Page Results in "Industry Standard" XML Format






Transforming XML Datagrams into HTML for Display

To return the same XML information in HTML instead of an alternative XML format, simply use a different XSLT stylesheet. Rather than producing elements like `<flight-list>` and `<flight>`, your stylesheet produces HTML elements like `<table>`, `<tr>`, and `<td>` instead. The result of the dynamically queried information would then look like the HTML page shown in [Figure 10-5](#). Instead of returning “raw” XML information, the XSQL Page leverages server-side XSLT transformation to format the information as HTML for delivery to the browser.

Figure 10-5 Using an Associated XSLT Stylesheet to Render HTML



The screenshot shows a web browser window with the address bar containing `http://localhost/examples/AvailableFlightsToday.xsql?City=JFK&xml-stylesheet=flight-display.xsl`. The browser displays a table with the following data:

Flight	Arrives
 VS 344	16:10
 LH 466	21:33
 UA 32	23:54

Similar to the syntax of the `flight-list.xsl` stylesheet, the `flight-display.xsl` stylesheet looks like a template HTML page, with `<xsl:for-each>`, `<xsl:value-of>` and attribute value templates like `{DUE}` to plug in the dynamic values from the underlying `<ROWSET>` and `<ROW>` structured XML query results.

```
<!-- XSLT Stylesheet to transform ROWSET/ROW results into HTML -->
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
  <head><link rel="stylesheet" type="text/css" href="flights.css" /></head>
  <body>
    <center><table border="0">
      <tr><th>Flight</th><th>Arrives</th></tr>
      <xsl:for-each select="ROWSET/ROW">
        <tr>
          <td>
            <table border="0" cellspacing="0" cellpadding="4">
              <tr>
                <td></td>
                <td width="180">
                  <xsl:value-of select="CARRIER"/>
                  <xsl:text> </xsl:text>
                  <xsl:value-of select="FLIGHTNUMBER"/>
                </td>
              </tr>
            </table>
          </td>
          <td align="center"><xsl:value-of select="DUE"/></td>
        </tr>
      </xsl:for-each>
    </table></center>
  </body>
</html>
```

Note: The stylesheet looks exactly like HTML, with one tiny difference. It is well-formed HTML. This means that each opening tag is properly closed (e.g. <td>...</td>) and that empty tags use the XML empty element syntax
 instead of just
.

You can see that by combining the power of:

- Parameterized SQL statements to select any information you need from our Oracle database,
- Industry-standard XML as a portable, interim data exchange format
- XSLT to transform XML-based "data pages" into any XML- or HTML-based format you need

you can achieve very interesting and useful results quickly. You will see in later sections that what you have seen above is just scratching the surface of what you can do using XSQL pages.

Note: For a detailed introduction to XSLT and a thorough tutorial on how to apply XSLT to many different Oracle database scenarios, see "Building Oracle XML Applications", by Steve Muench, from O'Reilly and Associates.

Setting Up and Using XSQL Pages in Your Environment

You can develop and use XSQL pages in a variety of ways. We start by describing the easiest way to get started, using Oracle JDeveloper, then cover the details you'll need to understand to use XSQL pages in your production environment.

Using XSQL Pages With Oracle JDeveloper

The easiest way to work with XSQL pages during development is to use Oracle JDeveloper. Versions 3.1 and higher of the JDeveloper IDE support color-coded syntax highlighting, XML syntax checking, and easy testing of your XSQL pages. In addition, the JDeveloper 3.2 release supports debugging XSQL pages and adds new wizards to help create XSQL actions.

To create an XSQL page in a JDeveloper project, you can:

- Click the plus icon at the top of the navigator to add a new or existing XSQL page to your project
- Select *File | New...* and select "XSQL" from the "Web Objects" tab of the gallery

To get assistance adding XSQL action elements like `<xsql:query>` to your XSQL page, place the cursor where you want the new element to go and either:

- Select *XSQL Element...* from the right mouse menu, or
- Select *Wizards | XSQL Element...* from the IDE menu.

The XSQL Element wizard takes you through the steps of selecting which XSQL action you want to use, and which attributes you need to provide.

To syntax-check an XSQL page template, you can select *Check XML Syntax...* at any time from the right-mouse menu in the navigator after selecting the name of the XSQL page you'd like to check. If there are any XML syntax errors, they will appear in the message view and your cursor will be brought to the first one.

To test an XSQL page, simply select the page in the navigator and choose *Run* from the right-mouse menu. JDeveloper automatically starts up a local Web-to-go web server, properly configured to run XSQL pages, and tests your page by launching your default browser with the appropriate URL to request the page. Once you've run the XSQL page, you can continue to make modifications to it in the IDE — as well as to any XSLT stylesheets with which it might be associated — and after saving the files in the IDE you can immediately refresh the browser to observe the effect of the changes.

Using JDeveloper, the "XSQL Runtime" library should be added to your project's library list so that the CLASSPATH is properly setup. The IDE adds this entry automatically when you go through the New Object gallery to create a new XSQL page, but you can also add it manually to the project by selecting *Project | Project Properties...* and clicking on the "Libraries" tab.

Setting the CLASSPATH Correctly in Your Production Environment

Outside of the JDeveloper environment, you need to make sure that the XSQL page processor engine is properly configured to run. Oracle comes with the XSQL Servlet pre-installed to the Oracle HTTP Server that accompanies the database, but using XSQL in any other environment, you'll need to ensure that the Java CLASSPATH is setup correctly.

There are three "entry points" to the XSQL page processor:

- `oracle.xml.xsql.XSQLServlet`, the servlet interface
- `oracle.xml.xsql.XSQLCommandLine`, the command line interface
- `oracle.xml.xsql.XSQLRequest`, the programmatic interface

Since all three of these interfaces, as well as the core XSQL engine itself, are written in Java, they are very portable and very simple to setup. The only setup requirements are to make sure the appropriate JAR files are in the CLASSPATH of the JavaVM that will be running processing the XSQL Pages. The JAR files include:

- `oraclexsql.jar`, the XSQL page processor
- `xmlparserv2.jar`, the Oracle XML Parser for Java v2
- `xsu12.jar`, the Oracle XML SQL utility
- `classes12.zip`, the Oracle JDBC driver

In addition, the *directory* where XSQL Page Processor's configuration file `XSQLConfig.xml` resides must also be listed as a directory in the CLASSPATH.

Putting all this together, if you have installed the XSQL distribution in `C:\xsql`, then your CLASSPATH would appear as follows:

```
C:\xsql\lib\classes12.zip;C:\xsql\lib\xmlparserv2.jar;
C:\xsql\lib\xsul2.jar;C:\xsql\lib\oraclexsql.jar;
directory_where_XSQLConfig.xml_resides
```

On Unix, if you extracted the XSQL distribution into your `/web` directory, the CLASSPATH would appear as follows:

```
/web/xsql/lib/classes12.zip:/web/xsql/lib/xmlparserv2.jar:
/web/xsql/lib/xsul2.jar:/web/xsql/lib/oraclexsql.jar:
directory_where_XSQLConfig.xml_resides
```

To use the XSQL Servlet, one additional setup step is required. You must associate the `.xsql` file extension with the XSQL Servlet's java class `oracle.xml.xsql.XSQLServlet`. How you set the CLASSPATH of the web server's servlet environment and how you associate a Servlet with a file extension are done differently for each web server. The XSQL Servlet's Release Notes contain detailed setup information for specific web servers you might want to use with XSQL Pages.

Setting Up the Connection Definitions

XSQL pages refer to database connections by using a “nickname” for the connection defined in the XSQL configuration file. Connection names are defined in the `<connectiondefs>` section of `XSQLConfig.xml` file like this:

```
<connectiondefs>
  <connection name="demo">
    <username>scott</username>
    <password>tiger</password>
    <dburl>jdbc:oracle:thin:@localhost:1521:testDB</dburl>
    <driver>oracle.jdbc.driver.OracleDriver</driver>
    <autocommit>true</autocommit>
  </connection>
  <connection name="lite">
    <username>system</username>
    <password>manager</password>
    <dburl>jdbc:Polite:POLite</dburl>
    <driver>oracle.lite.poljdbc.POLJDBCdriver</driver>
  </connection>
</connectiondefs>
```

For each connection, you can specify five pieces of information:

1. `<username>`
2. `<password>`
3. `<dburl>`, the JDBC connection string
4. `<driver>`, the fully-qualified class name of the JDBC driver to use
5. `<autocommit>`, optionally forces the autocommit to `true` or `false`

If the `<autocommit>` element is omitted, then the XSQL page processor will use the JDBC driver's default setting of the AutoCommit flag.

Any number of `<connection>` elements can be placed in this file to define the connections you need. An individual XSQL page refers to the connection it wants to use by putting a `connection="xxx"` attribute on the top-level element in the page (also called the "document element").

Note: For security reasons, when installing XSQL Servlet on your production web server, make sure the `XSQLConfig.xml` file does *not* reside in a directory that is part of the web server's virtual directory hierarchy. Failure to take this precaution risks exposing your configuration information over the web.

Using the XSQL Command Line Utility

Often the content of a dynamic page will be based on data that is not frequently changing in your environment. To optimize performance of your web publishing, you can use operating system facilities to schedule offline processing of your XSQL pages, leaving the processed results to be served statically by your web server.

You can process any XSQL page from the command line using the XSQL command line utility. The syntax is:

```
$ java oracle.xml.xsql.XSQLCommandLine xsqlpage [outfile] [param1=value1 ...]
```

If an *outfile* is specified, the result of processing *xsqlpage* is written to it, otherwise the result goes to standard out. Any number of parameters can be passed to the XSQL page processor and are available for reference by the XSQL page being processed as part of the request. However, the following parameter names are recognized by the command line utility and have a pre-defined behavior:

- `xml-stylesheet=stylesheetURL`

Provides the relative or absolute URL for a stylesheet to use for the request. Also can be set to the string `none` to suppress XSLT stylesheet processing for debugging purposes.

- `posted-xml=XMLDocumentURL`

Provides the relative or absolute URL of an XML resource to treat as if it were posted as part of the request.

- `useragent=UserAgentString`

Used to simulate a particular HTTP User-Agent string from the command line so that an appropriate stylesheet for that User-Agent type will be selected as part of command-line processing of the page.

The `*/xdk/java/xsql/bin` directory contains a platform-specific command script to automate invoking the XSQL command line utility. This script sets up the Java runtime to run `oracle.xml.xsql.XSQLCommandLine` class.

Overview of All XSQL Pages Capabilities

So far we've only seen a single XSQL action element, the `<xsql:query>` action. This is by far the most popular action, but it is not the only one that comes built-in to the XSQL Pages framework. We explore the full set of functionality that you can exploit in your XSQL pages in the following sections.

Using All of the Core Built-in Actions

This section provides a list of the core built-in actions, including a brief description of what each action does, and a listing of all required and optional attributes that each supports.

The `<xsql:query>` Action

The `<xsql:query>` action element executes a SQL select statement and includes a canonical XML representation of the query's result set in the data page. This action requires a database connection to be provided by supplying a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

The syntax for the action is:

```
<xsql:query>
  SELECT Statement
</xsql:query>
```

Any legal SQL select statement is allowed. If the select statement produces no rows, a "fallback" query can be provided by including a nested `<xsql:no-rows-query>` element like this:

```
<xsql:query>
  SELECT Statement
  <xsql:no-rows-query>
    SELECT Statement to use if outer query returns no rows
  </xsql:no-rows-query>
</xsql:query>
```

An `<xsql:no-rows-query>` element can *itself* contain nested `<xsql:no-rows-query>` elements to any level of nesting. The options available on the `<xsql:no-rows-query>` are identical to those available on the `<xsql:query>` action element.

By default, the XML produced by a query will reflect the column structure of its resultset, with element names matching the names of the columns. Columns in the result with nested structure like:

- Object Types
- Collection Types
- CURSOR Expressions

produce nested elements that reflect this structure. The result of a typical query containing different types of columns and returning one row might look like this:

```
<ROWSET>
  <ROW id="1">
    <VARCHARCOL>Value</VARCHARCOL>
    <NUMBERCOL>12345</NUMBERCOL>
    <DATECOL>12/10/2001 10:13:22</DATECOL>
    <OBJECTCOL>
      <ATTR1>Value</ATTR1>
      <ATTR2>Value</ATTR2>
    </OBJECTCOL>
    <COLLECTIONCOL>
      <COLLECTIONCOL_ITEM>
        <ATTR1>Value</ATTR1>
        <ATTR2>Value</ATTR2>
      </COLLECTIONCOL_ITEM>
      <COLLECTIONCOL_ITEM>
        <ATTR1>Value</ATTR1>
        <ATTR2>Value</ATTR2>
      </COLLECTIONCOL_ITEM>
    </COLLECTIONCOL>
  </ROW>
</ROWSET>
```

```

        </COLLECTIONCOL_ITEM>
    </COLLECTIONCOL>
    <CURSORCOL>
        <CURSORCOL_ROW>
            <COL1>Value1</COL1>
            <COL2>Value2</COL2>
        </CURSORCOR_ROW>
    </CURSORCOL>
</ROW>
</ROWSET>

```

A `<ROW>` element will repeat for each row in the result set. Your query can use standard SQL column aliasing to rename the columns in the result, and in doing so effectively rename the XML elements that are produced as well. Note that such column aliasing is *required* for columns whose names would otherwise be an illegal name for an XML element.

For example, an `<xsql:query>` action like this:

```
<xsql:query>SELECT TO_CHAR(hiredate, 'DD-MON') FROM EMP</xsql:query>
```

would produce an error because the default column name for the calculated expression will be an illegal XML element name. You can fix the problem with column aliasing like this:

```
<xsql:query>SELECT TO_CHAR(hiredate, 'DD-MON') as hiredate FROM EMP</xsql:query>
```

The optional attributes listed in [Table 10–1](#) can be supplied to control various aspects of the data retrieved and the XML produced by the `<xsql:query>` action.

Table 10–1 Attributes for `<xsql:query>`

Attribute Name	Description
<code>bind-params = "string"</code>	Ordered, space-separated list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
<code>date-format = "string"</code>	Date format mask to use for formatted date column/attribute values in XML being queried. Valid values are those documented for the <code>java.text.SimpleDateFormat</code> class.
<code>error-statement = "boolean"</code>	If set to <code>no</code> , suppresses the inclusion of the offending SQL statement in any <code><xsql-error></code> element generated. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>yes</code> .

Table 10–1 Attributes for <xsql:query>

Attribute Name	Description
fetch-size = "integer"	Number of records to fetch in each round-trip to the database. If not set, the default value is used as specified by the <code>/XSQLConfig/processor/default-fetch-size</code> configuration setting in <code>XSQLConfig.xml</code>
id-attribute = "string"	XML attribute name to use instead of the default <code>num</code> attribute for uniquely identifying each row in the result set. If the value of this attribute is the empty string, the row id attribute is suppressed.
id-attribute-column = "string"	Case-sensitive name of the column in the result set whose value should be used in each row as the value of the row id attribute. The default is to use the row count as the value of the row id attribute.
include-schema = "boolean"	If set to <code>yes</code> , includes an inline XML schema that describes the structure of the result set. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>no</code> .
max-rows = "integer"	Maximum number of rows to fetch, after optionally skipping the number of rows indicated by the <code>skip-rows</code> attribute. If not specified, default is to fetch all rows.
null-indicator = "boolean"	Indicates whether to signal that a column's value is NULL by including the <code>NULL="Y"</code> attribute on the element for the column. By default, columns with NULL values are omitted from the output. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>no</code> .
row-element = "string"	XML element name to use instead of the default <code><ROW></code> element name for the entire rowset of query results. Set to the empty string to suppress generating a containing <code><ROW></code> element for each row in the result set.
rowset-element = "string"	XML element name to use instead of the default <code><ROWSET></code> element name for the entire rowset of query results. Set to the empty string to suppress generating a containing <code><ROWSET></code> element.
skip-rows = "integer"	Number of rows to skip before fetching rows from the result set. Can be combined with <code>max-rows</code> for stateless paging through query results.
tag-case = "string"	Valid values are <code>lower</code> and <code>upper</code> . If not specified, the default is to use the case of column names as specified in the query as corresponding XML element names.

The <xsql:dml> Action

You can use the <xsql:dml> action to perform any DML or DDL operation, as well as any PL/SQL block. This action requires a database connection to be provided by supplying a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

The syntax for the action is:

```
<xsql:dml>
  DML Statement or DDL Statement or PL/SQL Block
</xsql:dml>
```

Table 10-2 lists the optional attributes that you can use on the <xsql:dml> action.

Table 10-2 Attributes for <xsql:dml>

Attribute Name	Description
<code>commit = "boolean"</code>	If set to <code>yes</code> , calls <code>commit</code> on the current connection after a successful execution of the DML statement. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>no</code> .
<code>bind-params = "string"</code>	Ordered, space-separated list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
<code>error-statement = "boolean"</code>	If set to <code>no</code> , suppresses the inclusion of the offending SQL statement in any <xsql-error> element generated. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>yes</code> .

The <xsql:ref-cursor-function> Action

The <xsql:ref-cursor-function> action allows you to include the XML results produced by a query whose result set is determined by executing a PL/SQL stored function. This action requires a database connection to be provided by supplying a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

By exploiting PL/SQL's dynamic SQL capabilities, the query can be dynamically and/or conditionally constructed by the function before a cursor handle to its result set is returned to the XSQL page processor. As its name implies, the return value of the function being invoked must be of type `REF CURSOR`.

The syntax of the action is:

```
<xsql:ref-cursor-function>
  [SCHEMA.] [PACKAGE.] FUNCTION_NAME(args);
</xsql:ref-cursor-function>
```

With the exception of the `fetch-size` attribute, the optional attributes available for the `<xsql:ref-cursor-function>` action are exactly the same as for the `<xsql:query>` action that are listed [Table 10-1](#).

For example, consider the PL/SQL package below:

```
CREATE OR REPLACE PACKAGE DynCursor IS
  TYPE ref_cursor IS REF CURSOR;
  FUNCTION DynamicQuery(id NUMBER) RETURN ref_cursor;
END;
CREATE OR REPLACE PACKAGE BODY DynCursor IS
  FUNCTION DynamicQuery(id NUMBER) RETURN ref_cursor IS
    the_cursor ref_cursor;
  BEGIN
    -- Conditionally return a dynamic query as a REF CURSOR
    IF id = 1 THEN
      OPEN the_cursor
        FOR 'SELECT empno, ename FROM EMP'; -- An EMP Query
    ELSE
      OPEN the_cursor
        FOR 'SELECT dname, deptno FROM DEPT'; -- A DEPT Query
    END IF;
    RETURN the_cursor;
  END;
END;
```

An `<xsql:ref-cursor-function>` can include the dynamic results of the REF CURSOR returned by this function by doing:

```
<xsql:ref-cursor-function>
  DynCursor.DynamicQuery(1);
</xsql:ref-cursor-function>
```

The `<xsql:include-owa>` Action

The `<xsql:include-owa>` action allows you to include XML content that has been generated by a database stored procedure. This action requires a database connection to be provided by supplying a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

The stored procedure uses the standard Oracle Web Agent (OWA) packages (`HTP` and `HTF`) to "print" the XML tags into the server-side page buffer, then the XSQL page processor fetches, parses, and includes the dynamically-produced XML content in the data page. The stored procedure must generate a well-formed XML page or an appropriate error is displayed.

The syntax for the action is:

```
<xsql:include-owa>
  PL/SQL Block invoking a procedure that uses the HTP and/or HTF packages
</xsql:include-owa>
```

Table 10–3 lists the optional attributes supported by this action.

Table 10–3 Attributes for `<xsql:include-owa>`

Attribute Name	Description
<code>bind-params = "string"</code>	Ordered, space-separated list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
<code>error-statement = "boolean"</code>	If set to <code>no</code> , suppresses the inclusion of the offending SQL statement in any <code><xsql-error></code> element generated. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>yes</code> .

Using Bind Variables

To parameterize the results of any of the above actions, you can use SQL bind variables. This allows your XSQL page template to produce different results based on the value of parameters passed in the request. To use a bind variable, simply include a question mark anywhere in the statement where bind variables are allowed by SQL. For example, your `<xsql:query>` action might contain the select statement:

```
SELECT s.ticker as "Symbol", s.last_traded_price as "Price"
  FROM latest_stocks s, customer_portfolio p
 WHERE p.customer_id = ?
       AND s.ticker = p.ticker
```

Using a question mark to create a bind-variable for the customer id. Whenever the SQL statement is executed in the page, parameter values are bound to the bind variable by specifying the `bind-params` attribute on the action element. Using the example above, we could create an XSQL page that binds the indicated bind variables to the value of the `custid` parameter in the page request like this:

```
<!-- CustomerPortfolio.xsql -->
<portfolio connection="prod" xmlns:xsql="urn:oracle-xsql">
  <xsql:query bind-params="custid">
    SELECT s.ticker as "Symbol", s.last_traded_price as "Price"
      FROM latest_stocks s, customer_portfolio p
     WHERE p.customer_id = ?
           AND s.ticker = p.ticker
  </xsql:query>
</portfolio>
```

The XML data for a particular customer's portfolio can then be requested by passing the customer id parameter in the request like this:

```
http://yourserver.com/fin/CustomerPortfolio.xsql?custid=1001
```

The value of the `bind-params` attribute is a space-separated list of parameter names whose left-to-right order indicates the positional bind variable to which its value will be bound in the statement. So, if your SQL statement has five question marks, then your `bind-params` attribute needs a space-separated list of five parameter names. If the same parameter value needs to be bound to several different occurrences of a question-mark-indicated bind variable, you simply repeat the name of the parameters in the value of the `bind-params` attribute at the appropriate position. Failure to include *exactly* as many parameter names in the `bind-params` attribute as there are question marks in the query, will result in an error when the page is executed.

Bind variables can be used in any action that expects a SQL statement. The following page gives additional examples:

```
<!-- CustomerPortfolio.xsql -->
<portfolio connection="prod" xmlns:xsql="urn:oracle-xsql">
  <xsql:dml commit="yes" bind-params="useridCookie">
    BEGIN log_user_hit(?); END;
  </xsql:dml>
  <current-prices>
    <xsql:query bind-params="custid">
      SELECT s.ticker as "Symbol", s.last_traded_price as "Price"
        FROM latest_stocks s, customer_portfolio p
       WHERE p.customer_id = ?
             AND s.ticker = p.ticker
    </xsql:query>
  </current-prices>
  <analysis>
    <xsql:include-owa bind-params="custid userCookie">
      BEGIN portfolio_analysis.historical_data(?,5 /* years */, ?); END;
```



```

    </xsql:include-owa>
  </analysis>
</portfolio>

```

Using Lexical Substitution Parameters

For any XSQL action element, you can substitute the value of any attribute, or the text of any contained SQL statement, by using a lexical substitution parameter. This allows you to parameterize how the actions behave as well as substitute parts of the SQL statements they perform. Lexical substitution parameters are referenced using the syntax `{@ParameterName}`.

The following example illustrates using two *lexical* substitution parameters, one which allows the maximum number of rows to be passed in as a parameter, and the other which controls the list of columns to ORDER BY.

```

<!-- DevOpenBugs.xsql -->
<open-bugs connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max-rows="{@max}" bind-params="dev prod">
    SELECT bugno, abstract, status
    FROM bug_table
    WHERE programmer_assigned = UPPER(?)
    AND product_id           = ?
    AND status < 80
    ORDER BY {@orderby}
  </xsql:query>
</open-bugs>

```

This example could then show the XML for a given developer's open bug list by requesting the URL:

```
http://yourserver.com/bug/DevOpenBugs.xsql?dev=smuench&prod=817
```

or using the XSQL Command Line Utility to request:

```
$ xsql DevOpenBugs.xsql dev=smuench prod=817
```

We close by noting that lexical parameters can also be used to parameterize the XSQL page connection, as well as parameterize the stylesheet that is used to process the page like this:

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="{@sheet}.xsl"?>
<!-- DevOpenBugs.xsql -->
<open-bugs connection="{@conn}" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max-rows="{@max}" bind-params="dev prod">
    SELECT bugno, abstract, status

```

```
FROM bug_table
WHERE programmer_assigned = UPPER(?)
AND product_id           = ?
AND status < 80
ORDER BY {@orderby}
</xsql:query>
</open-bugs>
```

Providing Default Values for Bind Variables and Parameters

It is often convenient to provide a default value for a bind variable or a substitution parameter directly in the page. This allows the page to be parameterized without requiring the requester to explicitly pass in all the values in each request.

To include a default value for a parameter, simply add an XML attribute of the same name as the parameter to the action element, or to any ancestor element. If a value for a given parameter is not included in the request, the XSQL page processor looks for an attribute by the same name on the current action element. If it doesn't find one, it keeps looking for such an attribute on each ancestor element of the current action element until it gets to the document element of the page.

As a simple example, the following page defaults the value of the `max` parameter to 10 for both `<xsql:query>` actions in the page:

```
<example max="10" connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max-rows="{@max}">SELECT * FROM TABLE1</xsql:query>
  <xsql:query max-rows="{@max}">SELECT * FROM TABLE2</xsql:query>
</example>
```

This example defaults the first query to have a `max` of 5, the second query to have a `max` of 7 and the third query to have a `max` of 10.

```
<example max="10" connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query max="5" max-rows="{@max}">SELECT * FROM TABLE1</xsql:query>
  <xsql:query max="7" max-rows="{@max}">SELECT * FROM TABLE2</xsql:query>
  <xsql:query max-rows="{@max}">SELECT * FROM TABLE3</xsql:query>
</example>
```

Of course, all of these defaults would be overridden if a value of `max` is supplied in the request like:

```
http://yourserver.com/example.xsql?max=3
```

Bind variables respect the same defaulting rules so a — not-very-useful, yet educational — page like this:

```
<example val="10" connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query tag-case="lower" bind-params="val val val">
    SELECT ? as somevalue
      FROM DUAL
     WHERE ? = ?
  </xsql:query>
</example>
```

Would return the XML datagram:

```
<example>
  <rowset>
    <row>
      <somevalue>10</somevalue>
    </row>
  </rowset>
</example>
```

if the page were requested without any parameters, while a request like:

```
http://yourserver.com/example.xsql?val=3
```

Would return:

```
<example>
  <rowset>
    <row>
      <somevalue>3</somevalue>
    </row>
  </rowset>
</example>
```

To illustrate an important point for bind variables, imagine removing the default value for the `val` parameter from the page by removing the `val` attribute like this:

```
<example connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:query tag-case="lower" bind-params="val val val">
    SELECT ? as somevalue
      FROM DUAL
     WHERE ? = ?
  </xsql:query>
</example>
```

Now a request for the page without supplying any parameters would return:

```
<example>
  <rowset/>
</example>
```

because a bind variable that is bound to a parameter with *neither* a default value *nor* a value supplied in the request will be bound to NULL, causing the WHERE clause in our example page above to return no rows.

Understanding the Different Kinds of Parameters

XSQL pages can make use of parameters supplied in the request, as well as page-private parameters whose names and values are determined by actions in the page. If an action encounters a reference to a parameter named *param* in either a *bind-params* attribute or in a lexical parameter reference, the value of the *param* parameter is resolved by using:

1. The value of the page-private parameter named *param*, if set, otherwise
2. The value of the request parameter named *param*, if supplied, otherwise
3. The default value provided by an attribute named *param* on the current action element or one of its ancestor elements, otherwise
4. The value NULL for bind variables and the empty string for lexical parameters

For XSQL pages that are processed by the XSQL Servlet over HTTP, two additional HTTP-specific type of parameters are available to be set and referenced. These are HTTP-Session-level variables and HTTP Cookies. For XSQL pages processed through the XSQL Servlet, the parameter value resolution scheme is augmented as follows. The value of a parameter *param* is resolved by using:

1. The value of the page-private parameter *param*, if set, otherwise
2. The value of the cookie named *param*, if set, otherwise
3. The value of the session variable named *param*, if set, otherwise
4. The value of the request parameter named *param*, if supplied, otherwise
5. The default value provided by an attribute named *param* on the current action element or one of its ancestor elements, otherwise
6. The value NULL for bind variables and the empty string for lexical parameters

The resolution order is arranged this way so that users cannot supply parameter values in a request to override parameters of the same name that have been set in the HTTP session — whose lifetime is the duration of the HTTP session and controlled by your web server — or set as cookies, which can be set to "live" across browser sessions.

The `<xsql:include-request-params>` Action

The `<xsql:include-request-params>` action allows you to include an XML representation of all parameters in the request in your datagram. This is useful if your associated XSLT stylesheet wants to refer to any of the request parameter values by using XPath expressions.

The syntax of the action is:

```
<xsql:include-request-params/>
```

The XML included will have the form:

```
<request>
  <parameters>
    <paramname>value1</paramname>
    <ParamName2>value2</ParamName2>
    :
  </parameters>
</request>
```

or the form:

```
<request>
  <parameters>
    <paramname>value1</paramname>
    <ParamName2>value2</ParamName2>
    :
  </parameters>
  <session>
    <sessVarName>value1</sessVarName>
    :
  </session>
  <cookies>
    <cookieName>value1</cookieName>
    :
  </cookies>
</request>
```

when processing pages through the XSQL Servlet.

This action has no required or optional attributes.

The `<xsql:include-param>` Action

The `<xsql:include-param>` action allows you to include an XML representation of a single parameter in your datagram. This is useful if your associated XSLT stylesheet wants to refer to the parameter's value by using an XPath expression.

The syntax of the action is:

```
<xsql:include-param name="paramname" />
```

This `name` attribute is required, and supplies the name of the parameter whose value you would like to include. This action has no optional attributes.

The XML included will have the form:

```
<paramname>value1</paramname>
```

The `<xsql:include-xml>` Action

The `<xsql:include-xml>` action includes the XML contents of a local or remote resource into your datagram. The resource is specified by URL.

The syntax for this action is:

```
<xsql:include-xml href="URL"/>
```

The URL can be an absolute, http-based URL to retrieve XML from another web site, or a relative URL to include XML from a file on the file system. The `href` attribute is required, and this action has no other optional attributes.

The `<xsql:set-page-param>` Action

The `<xsql:set-page-param>` action sets a page-private parameter to a value. The value can be supplied by a combination of static text and other parameter values, or alternatively from the result of a SQL select statement.

The syntax for this action is:

```
<xsql:set-page-param name="paramname" value="value"/>
```

or

```
<xsql:set-page-param name="paramname">  
  SQL select statement  
</xsql:set-page-param>
```

If you use the SQL statement option, a single row is fetched from the result set and the parameter is assigned the value of the *first* column. This use requires a database

connection to be provided by supplying a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

The `name` attribute is required. The `value` attribute and the contained SQL statement are mutually exclusive. If one is supplied, the other must not be.

Table 10–4 lists the attributes supported by this action. Attributes in bold are required.

Table 10–4 Attributes for `<xsql:set-page-param>`

Attribute Name	Description
<code>name = "string"</code>	Name of the page-private parameter whose value you want to set.
<code>bind-params = "string"</code>	Ordered, space-separated list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
<code>ignore-empty-value = "boolean"</code>	Indicates whether the page-level parameter assignment should be ignored if the value to which it is being assigned is an empty string. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>no</code> .

The `<xsql:set-session-param>` Action

The `<xsql:set-session-param>` action sets an HTTP session-level parameter to a value. The value of the session-level parameter remains for the lifetime of the current browser user's HTTP session, which is controlled by the web server. The value can be supplied by a combination of static text and other parameter values, or alternatively from the result of a SQL select statement.

Since this feature is specific to Java Servlets, this action is only effective if the XSQL page in which it appears is being processed by the XSQL Servlet. If this action is encountered in an XSQL page being processed by the XSQL command line utility or the `XSQLRequest` programmatic API, this action is a no-op.

The syntax for this action is:

```
<xsql:set-session-param name="paramname" value="value"/>
```

or

```
<xsql:set-session-param name="paramname">
  SQL select statement
</xsql:set-session-param>
```

If you use the SQL statement option, a single row is fetched from the result set and the parameter is assigned the value of the *first* column. This use requires a database connection to be provided by supplying a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

The `name` attribute is required. The `value` attribute and the contained SQL statement are mutually exclusive. If one is supplied, the other must not be.

[Table 10-5](#) lists the optional attributes supported by this action.

Table 10-5 Attributes for `<xsql:set-session-param>`

Attribute Name	Description
<code>name = "string"</code>	Name of the session-level variable whose value you want to set.
<code>bind-params = "string"</code>	Ordered, space-separated list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
<code>ignore-empty-value = "boolean"</code>	Indicates whether the page-level parameter assignment should be ignored if the value to which it is being assigned is an empty string. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>no</code> .
<code>only-if-unset = "boolean"</code>	Indicates whether the session variable assignment should only occur when the session variable currently does not exist. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>no</code> .

The `<xsql:set-cookie>` Action

The `<xsql:set-cookie>` action sets an HTTP cookie to a value. By default, the value of the cookie remains for the lifetime of the current browser, but its lifetime can be changed by supplying the optional `max-age` attribute. The value can be supplied by a combination of static text and other parameter values, or alternatively from the result of a SQL select statement.

Since this feature is specific to the HTTP protocol, this action is only effective if the XSQL page in which it appears is being processed by the XSQL Servlet. If this action is encountered in an XSQL page being processed by the XSQL command line utility or the `XSQLRequest` programmatic API, this action is a no-op.

The syntax for this action is:

```
<xsql:set-cookie name="paramname" value="value" />
```

or


```
<xsql:set-cookie name="paramname">
  SQL select statement
</xsql:set-cookie>
```

If you use the SQL statement option, a single row is fetched from the result set and the parameter is assigned the value of the *first* column. This use requires a database connection to be provided by supplying a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

The `name` attribute is required. The `value` attribute and the contained SQL statement are mutually exclusive. If one is supplied, the other must not be.

Table 10–6 lists the optional attributes supported by this action.

Table 10–6 Attributes for <xsql:set-cookie>

Attribute Name	Description
<code>name = "string"</code>	Name of the cookie whose value you want to set.
<code>bind-params = "string"</code>	Ordered, space-separated list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
<code>domain = "string"</code>	Domain in which cookie value is valid and readable. If domain is not set explicitly, then it defaults to the fully-qualified hostname (e.g. <code>bigserver.yourcompany.com</code>) of the document creating the cookie.
<code>ignore-empty-value = "boolean"</code>	Indicates whether the page-level parameter assignment should be ignored if the value to which it is being assigned is an empty string. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>no</code> .
<code>max-age = "integer"</code>	Sets the maximum age of the cookie in <i>seconds</i> . Default is to set the cookie to expire when users current browser session terminates.
<code>only-if-unset = "boolean"</code>	Indicates whether the cookie assignment should only occur when the cookie currently does not exist. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>no</code> .
<code>path = "string"</code>	Relative URL path within domain in which cookie value is valid and readable. If path is not set explicitly, then it defaults to the URL path of the document creating the cookie.

The <xsql:set-stylesheet-param> Action

The <xsql:set-stylesheet-param> action sets a top-level XSLT stylesheet parameter to a value. The value can be supplied by a combination of static text and other parameter values, or alternatively from the result of a SQL select statement. The stylesheet parameter will be set on any stylesheet used during the processing of the current page.

The syntax for this action is:

```
<xsql:set-stylesheet-param name="paramname" value="value"/>
```

or

```
<xsql:set-stylesheet-param name="paramname">
  SQL select statement
</xsql:set-stylesheet-param>
```

If you use the SQL statement option, a single row is fetched from the result set and the parameter is assigned the value of the *first* column. This use requires a database connection to be provided by supplying a `connection="connname"` attribute on the document element of the XSQL page in which it appears.

The `name` attribute is required. The `value` attribute and the contained SQL statement are mutually exclusive. If one is supplied, the other must not be.

Table 10-7 lists the optional attributes supported by this action.

Table 10-7 Attributes for <xsql:set-stylesheet-param>

Attribute Name	Description
<code>name = "string"</code>	Name of the top-level stylesheet parameter whose value you want to set.
<code>bind-params = "string"</code>	Ordered, space-separated list of one or more XSQL parameter names whose values will be used to bind to the JDBC bind variable in the appropriate sequential position in the SQL statement.
<code>ignore-empty-value = "boolean"</code>	Indicates whether the page-level parameter assignment should be ignored if the value to which it is being assigned is an empty string. Valid values are <code>yes</code> and <code>no</code> . The default value is <code>no</code> .

Aggregating Information Using <xsql:include-xsql>

The <xsql:include-xsql> action makes it very easy to include the results of one XSQL page into another page. This allows you to easily aggregate content from a

page that you've already built and repurpose it. The examples below illustrate two of the most common uses of `<xsql:include-xsql>`.

Assume you have an XSQL page that lists discussion forum categories:

```
<!-- Categories.xsql -->
<xsql:query connection="forum" xmlns:xsql="urn:oracle-xsql">
  SELECT name
  FROM categories
  ORDER BY name
</xsql:query>
```

You can include the results of this page into a page that lists the ten most recent topics in the current forum like this:

```
<!-- TopTenTopics.xsql -->
<top-ten-topics connection="forum" xmlns:xsql="urn:oracle-xsql">
  <topics>
    <xsql:query max-rows="10">
      SELECT subject FROM topics ORDER BY last_modified DESC
    </xsql:query>
  </topics>
  <categories>
    <xsql:include-xsql href="Categories.xsql"/>
  </categories>
</top-ten-topics>
```

You can use `<xsql:include-xsql>` to include an existing page to apply an XSLT stylesheet to it as well. So, if we have two different XSLT stylesheets:

- `cats-as-html.xml`, which renders the topics in HTML, and
- `cats-as-wml.xml`, which renders the topics in WML

Then one approach for catering to two different types of devices is to create different XSQL pages for each device. We can create:

```
<?xml version="1.0"?>
<!-- HTMLCategories.xsql -->
<?xml-stylesheet type="text/xsl" href="cats-as-html.xml"?>
<xsql:include-xsql href="Categories.xsql" xmlns:xsql="urn:oracle-xsql"/>
```

which aggregates `Categories.xsql` and applies the `cats-as-html.xml` stylesheet, and another page:

```
<?xml version="1.0"?>
<!-- WMLCategories.xsql -->
```

```
<?xml-stylesheet type="text/xsl" href="cats-as-html.xsl"?>
<xsql:include-xsql href="Categories.xsql" xmlns:xsql="urn:oracle-xsql"/>
```

which aggregates `Categories.xsql` and applies the `cats-as-wml.xsl` stylesheet for delivering to wireless devices. In this way, we've repurposed the reusable `Categories.xsql` page content in two different ways.

If the page being aggregated contains an `<?xml-stylesheet?>` processing instruction, then that stylesheet is applied before the result is aggregated, so using `<xsql:include-xsql>` you can also easily chain the application of XSLT stylesheets together.

When one XSQL page aggregates another page's content using `<xsql:include-xsql>` all of the request-level parameters are visible to the "nested" page. For pages processed by the XSQL Servlet, this also includes session-level parameters and cookies, too. As you would expect, none of the aggregating page's *page-private* parameters are visible to the nested page.

[Table 10-8](#) lists the attributes supported by this action. Required attributes are in bold.

Table 10-8 Attributes for `<xsql:include-xsql>`

Attribute Name	Description
href = "string"	Relative or absolute URL of XSQL page to be included.
reparse = "boolean"	Indicates whether output of included XSQL page should be <i>reparsed</i> before it is included. Useful if included XSQL page is selecting the <i>text</i> of an XML document fragment that the including page wants to treat as elements. Valid values are <i>yes</i> and <i>no</i> . The default value is <i>no</i> .

Handling Posted Information

In addition to simplifying the assembly and transformation of XML content, the XSQL Pages framework makes it easy to handle posted XML content as well. Built-in actions simplify the handling of posted information from both XML document and HTML forms, and allow that information to be posted directly into a database table using the underlying facilities of the Oracle XML SQL Utility.

The XML SQL Utility provides the ability to data database inserts, updates, and deletes based on the content of an XML document in "canonical" form with respect to a target table or view. For a given database table, the "canonical" XML form of its data is given by one row of XML output from a `SELECT * FROM tablename` query against it. Given an XML document in this canonical form, the XML SQL

Utility can automate the insert, update, and/or delete for you. By combining the XML SQL Utility with an XSLT transformation, you can transform XML in any format into the canonical format expected by a given table, and then ask the XML SQL Utility to insert, update, delete the resulting "canonical" XML for you.

The following built-in XSQL actions make exploiting this capability easy from within your XSQL pages:

- `<xsql:insert-request>`

Insert the optionally transformed XML document that was posted in the request into a table. [Table 10-9](#) lists the required and optional attributes supported by this action.

- `<xsql:update-request>`

Update the optionally transformed XML document that was posted in the request into a table or view. [Table 10-10](#) lists the required and optional attributes supported by this action.

- `<xsql:delete-request>`

Delete the optionally transformed XML document that was posted in the request from a table or view. [Table 10-11](#) lists the required and optional attributes supported by this action.

- `<xsql:insert-param>`

Insert the optionally transformed XML document that was posted as the value of a request parameter into a table or view. [Table 10-12](#) lists the required and optional attributes supported by this action.

If you target a database view with your insert, then you can create `INSTEAD OF INSERT` triggers on the view to further automate the handling of the posted information. For example, an `INSTEAD OF INSERT` trigger on a view could use PL/SQL to check for the existence of a record and intelligently choose whether to do an `INSERT` or an `UPDATE` depending on the result of this check.

Table 10-9 Attributes for `<xsql:insert-request>`

Attribute Name	Description
<code>table = "string"</code>	Name of the table, view, or synonym to use for inserting the XML information.
<code>transform = "URL"</code>	Relative or absolute URL of the XSLT transformation to use to transform the document to be inserted into canonical ROWSET/ROW format.

Table 10–9 Attributes for <xsql:insert-request>

Attribute Name	Description
columns = "string"	Space-separated or comma-separated list of one or more column names whose values will be inserted. If supplied, then only these columns will be inserted. If not supplied, all columns will be inserted, with NULL values for columns whose values do not appear in the XML document.
commit-batch-size = "integer"	If a positive, non-zero number N is specified, then after each batch of N inserted records, a commit will be issued. Default batch size is zero (0) if not specified, meaning not to commit interim batches.
date-format = "string"	Date format mask to use for interpreting date field values in XML being inserted. Valid values are those documented for the <code>java.text.SimpleDateFormat</code> class.

Table 10–10 Attributes for <xsql:update-request>

Attribute Name	Description
table = "string"	Name of the table, view, or synonym to use for inserting the XML information.
key-columns = "string"	Space-separated or comma-separated list of one or more column names whose values in the posted XML document will be used to identify the existing rows to update.
transform = "URL"	Relative or absolute URL of the XSLT transformation to use to transform the document to be inserted into canonical ROWSET/ROW format.
columns = "string"	Space-separated or comma-separated list of one or more column names whose values will be updated. If supplied, then only these columns will be updated. If not supplied, all columns will be updated, with NULL values for columns whose values do not appear in the XML document.
commit-batch-size = "integer"	If a positive, non-zero number N is specified, then after each batch of N inserted records, a commit will be issued. Default batch size is zero (0) if not specified, meaning not to commit interim batches.
date-format = "string"	Date format mask to use for interpreting date field values in XML being inserted. Valid values are those documented for the <code>java.text.SimpleDateFormat</code> class.

Table 10–11 Attributes for <xsql:delete-request>

Attribute Name	Description
<code>table = "string"</code>	Name of the table, view, or synonym to use for inserting the XML information.
<code>key-columns = "string"</code>	Space-separated or comma-separated list of one or more column names whose values in the posted XML document will be used to identify the existing rows to update.
<code>transform = "URL"</code>	Relative or absolute URL of the XSLT transformation to use to transform the document to be inserted into canonical ROWSET/ROW format.
<code>commit-batch-size = "integer"</code>	If a positive, non-zero number N is specified, then after each batch of N inserted records, a commit will be issued. Default batch size is zero (0) if not specified, meaning not to commit interim batches.

Table 10–12 Attributes for <xsql:insert-param>

Attribute Name	Description
<code>name = "string"</code>	Name of the parameter whose value contains XML to be inserted.
<code>table = "string"</code>	Name of the table, view, or synonym to use for inserting the XML information.
<code>transform = "URL"</code>	Relative or absolute URL of the XSLT transformation to use to transform the document to be inserted into canonical ROWSET/ROW format.
<code>columns = "string"</code>	Space-separated or comma-separated list of one or more column names whose values will be inserted. If supplied, then only these columns will be inserted. If not supplied, all columns will be inserted, with NULL values for columns whose values do not appear in the XML document.
<code>commit-batch-size = "integer"</code>	If a positive, non-zero number N is specified, then after each batch of N inserted records, a commit will be issued. Default batch size is zero (0) if not specified, meaning not to commit interim batches.

Table 10–12 Attributes for <xsql:insert-param>

Attribute Name	Description
date-format = "string"	Date format mask to use for interpreting date field values in XML being inserted. Valid values are those documented for the <code>java.text.SimpleDateFormat</code> class.

Understanding Different XML Posting Options

There are three different ways that the XSQL pages framework can handle posted information.

1. A client program can send an HTTP POST message that targets an XSQL page, whose request body contains an XML document and whose HTTP header reports a `ContentType` of "text/xml".

In this case, you can use the `<xsql:insert-request>`, `<xsql:update-request>`, or the `<xsql:delete-request>` action and the content of the posted XML will be insert, updated, or deleted in the target table as indicated. If you transform the posted XML document using an XSLT transformation, the posted XML document is the source document for this transformation.

2. A client program can send an HTTP GET request for an XSQL page, one of whose parameters contains an XML document.

In this case, you can use the `<xsql:insert-param>` action and the content of the posted XML parameter value will be inserted in the target table as indicated. If you transform the posted XML document using an XSLT transformation, the XML document in the parameter value is the source document for this transformation.

3. A browser can submit an HTML form with `method="POST"` whose action targets an XSQL page. In this case, by convention the browser sends an HTTP POST message whose request body contains an encoded version of all of the HTML form's fields and their values with a `ContentType` of "application/x-www-form-urlencoded"

In this case, there request does not contain an XML document, but instead an encoded version of the form parameters. However, to make all three of these cases uniform, the XSQL page processor will (on demand) materialize an XML document from the set of form parameters, session variables, and cookies contained in the request. Your XSLT transformation then transforms this dynamically-materialized XML document into canonical form for

insert, update, or delete using `<xsql:insert>`,
`<xsql:update-request>`, or `<xsql:delete-request>` respectively.

When working with posted HTML forms, the dynamically materialized XML document will have the following form:

```
<request>
  <parameters>
    <firstparamname>firstparamvalue</firstparamname>
    :
    <lastparamname>lastparamvalue</lastparamname>
  </parameters>
  <session>
    <firstparamname>firstsessionparamvalue</firstparamname>
    :
    <lastparamname>lastsessionparamvalue</lastparamname>
  </session>
  <cookies>
    <firstcookie>firstcookievalue</firstcookiename>
    :
    <lastcookie>firstcookievalue</lastcookiename>
  </cookies>
</request>
```

If multiple parameters are posted with the same name, then they will automatically be "row-ified" to make subsequent processing easier. This means, for example, that a request which posts or includes the following parameters:

- id = 101
- name = Steve
- id = 102
- name = Sita
- operation = update

Will create a "row-ified" set of parameters like:

```
<request>
  <parameters>
    <row>
      <id>101</id>
      <name>Steve</name>
    </row>
    <row>
      <id>102</id>
```

```
<name>Sita</name>
</row>
<operation>update</operation>
</parameters>
:
</request>
```

Since you will need to provide an XSLT stylesheet that transforms this materialized XML document containing the request parameters into canonical format for your target table, it might be useful to build yourself an XSQL page like this:

```
<!--
| ShowRequestDocument.xsql
| Show Materialized XML Document for an HTML Form
+-->
<xsql:include-request-params xmlns:xsql="urn:oracle-xsql"/>
```

With this page in place, you can temporarily modify your HTML form to post to the `ShowRequestDocument.xsql` page, and in the browser you will see the "raw" XML for the materialized XML request document which you can save out and use to develop the XSLT transformation.

Using Custom XSQL Action Handlers

When you need to perform tasks that are not handled by the built-in action handlers, the XSQL Pages framework allows custom actions to be invoked to do virtually any kind of job you need done as part of page processing. Custom actions can supply arbitrary XML content to the data page and perform arbitrary processing. See [Writing Custom XSQL Action Handlers](#) later in this chapter for more details on writing custom action handlers in Java. Here we explore how to make use of a custom action handler, once it's already created.

To invoke a custom action handler, use the built-in `<xsql:action>` action element. It has a single, required attribute named `handler` whose value is the fully-qualified Java class name of the action you want to invoke. The class must implement the `oracle.xml.xsql.XSQLActionHandler` interface. For example:

```
<xsql:action handler="yourpackage.YourCustomHandler"/>
```

Any number of additional attribute can be supplied to the handler in the normal way. For example, if the `yourpackage.YourCustomHandler` is expecting a attributes named `param1` and `param2`, you use the syntax:

```
<xsql:action handler="yourpackage.YourCustomHandler" param1="xxx" param2="yyy"/>
```

Some action handlers, perhaps in addition to attributes, may expect text content or element content to appear inside the `<xsql:action>` element. If this is the case, simply use the expected syntax like:

```
<xsql:action handler="yourpackage.YourCustomHandler" param1="xxx" param2="yyy">  
  Some Text Goes Here  
</xsql:action>
```

or this:

```
<xsql:action handler="yourpackage.YourCustomHandler" param1="xxx" param2="yyy">  
  <some>  
    <other/>  
    <elements/>  
    <here/>  
  </some>  
</xsql:action>
```

Description of XSQL Servlet Examples

Figure 10–13 lists the XSQL Servlet example applications supplied with the software in the `./demo` directory.

Table 10–13 XSQL Servlet Examples

Demonstration Name	Description
Hello World <code>./demo/helloworld</code>	Simplest possible XSQL page.
Do You XML Site <code>./demo/doyouxml</code>	<p>XSQL page shows how a to build a data-driven web site with an XSQL page. Uses SQL, XSQL-substitution variables in queries, and XSLT to format.</p> <p>Uses substitution parameters in SQL statements in <code><xsql:query></code> tags, and in attributes to <code><xsql:query></code> tags, to control for example how many records to display, or to skip, for paging through query results.</p>
Employee Page <code>./demo/emp</code>	<p>XSQL page displays XML data from EMP table, using XSQL page parameters to control employees and data sorting.</p> <p>Uses an associated XSLT Stylesheet to format results as HTML version of <code>emp.xsql</code> page. This is the form action hence you can fine tune your search criteria.</p>
Insurance Claim Page <code>./demo/insclaim</code>	<p>Shows sample queries over a structured, Insurance Claim object view.</p> <p><code>insclaim.sql</code> sets up the INSURANCE_CLAIM_VIEW object view and populates it with sample data.</p>
Invalid Classes Page <code>./demo/classerr</code>	<p>XSQL Page uses <code>invalidclasses.xsl</code> to format a “live” list of current Java class compilation errors in your schema. The <code>.sql</code> script sets up XSQLJavaClassesView object view for the demo. Master/detail information from object view is formatted into HTML by the <code>invalidclasses.xsl</code> stylesheet in the server.</p>
Airport Code Validation <code>./demo/airport</code>	<p>XSQL page returns a “datagram” of information about airports based on their three-letter codes. Uses <code><xsql:no-rows-query></code> as alternative queries when initial queries return no rows. After attempting to match the airport code passed in, the XSQL page tries a fuzzy match based on the airport description.</p> <p><code>airport.htm</code> page demonstrates how to use the XML results of <code>airport.xsql</code> page from a web page using JavaScript to exploit built-in XML Document Object Model (DOM) functionality in Internet Explorer 5.0.</p> <p>When you enter the three-letter airport code on the web page, a JavaScript fetches the XML datagram from XSQL Servlet over the web corresponding to the code you entered. If the return indicates no match, the program collects a “picklist” of possible matches based on information returned in the XML “datagram” from XSQL Servlet</p>

Table 10–13 XSQL Servlet Examples(Cont.)

Demonstration Name	Description
Airport Code Display ./demo/airport	Demonstrates using the same XSQL page as the Airport Code Validation example but supplying an XSLT Stylesheet name in the request. This causes the airport information to be formatted as an HTML form instead of being returned as raw XML.
Emp/Dept Object Demo ./demo/empdept	<p>How to use an object view to group master/detail information from two existing "flat" tables like EMP and DEPT. <code>empdeptobjs.sql</code> script creates the object view and INSTEAD OF INSERT triggers, allowing the use of master/detail view as an insert target of <code>xsq:insert-request</code>.</p> <p><code>empdept.xsl</code> stylesheet illustrates an example of the "simple form" of an XSLT stylesheet that can look just like an HTML page without the extra <code>xsl:stylesheet</code> or <code>xsl:transform</code> at the top. Part of XSLT 1.0 specification called using a Literal Result Element as Stylesheet.</p> <p>Shows how to generate an HTML page that includes the <code><link rel="stylesheet"></code> to allow the generated HTML to fully leverage CSS for centralized HTML style information, found in the <code>coolcolors.css</code> file.</p>
Adhoc Query Visualization ./demo/adhocsql	<p>Shows how to pass an SQL query and XSLT Stylesheet to use as parameters to the server.</p> <p>NOTE: <i>Deploying this demo page to your production environment should be given particular consideration because it allows the results of any SQL query in XML format over the Web that your SCOTT user account has access to.</i></p>
XML Document Demo ./demo/document	<p>How to insert XML documents into relational tables.</p> <p><code>docdemo.sql</code> script creates a user-defined type called <code>XMLDOCFRAG</code> containing an attribute of type <code>CLOB</code>.</p> <ul style="list-style-type: none"> ■ Insert the text of the document in <code>./xsql/demo/xml99.xml</code> and provide the name <code>xml99.xsl</code> as the stylesheet ■ Insert the text of the document in <code>./xsql/demo/JDevRelNotes.xml</code> with the stylesheet <code>relnotes.xsl</code>. <p><code>docstyle.xsql</code> page illustrates an example of the <code><xsql:include-xsql></code> action element to include the output of the <code>doc.xsql</code> page into its own page before transforming the final output using a client-supplied stylesheet name.</p> <p>XML Document demo uses client-side XML features of Internet Explorer 5.0 to check the document for well-formedness before it is posted to the server.</p>

Table 10–13 XSQL Servlet Examples(Cont.)

Demonstration Name	Description
XML Insert Request Demo ./demo/insertxml	<p>Posts XML from a client to an XSQL Page that inserts the posted XML information into a database table using the <code><xsql:insert-request></code> action element.</p> <p>The demo accepts XML documents in the moreover.com XML-based news format. The program posting the XML is a client-side web page using Internet Explorer 5.0 and the XMLHttpRequest object from JavaScript.</p> <p>The source for <code>insertnewsstory.xsql</code> page, specifies a table name and XSLT Transform name.</p> <p><code>moreover-to-newsstory.xsl</code> stylesheet transforms the incoming XML into canonical format that OracleXMLSave utility can insert. Copy and paste the example <code><article></code> element several times within the <code><moreovernews></code> element to insert several new articles in one shot.</p> <p><code>newsstory.sql</code> shows how INSTEAD OF triggers can be used on the database views into which you ask XSQL Pages to insert to the data to customize how incoming data is handled, default primary key values,....</p>
SVG Demo ./demo/svg	<p><code>deptlist.xsql</code> page displays a simple list of departments with hyperlinks to <code>SalChart.xsql</code> page.</p> <p><code>SalChart.xsql</code> page queries employees for a given department passed in as a parameter and uses the <code>SalChart.xsql</code> stylesheet to format the result into a Scalable Vector Graphics drawing, a bar chart comparing salaries of the employees in that department.</p>
PDF Demo ./demo/fop	<p><code>emptable.xsql</code> page displays a simple list of employees. The <code>emptable.xsl</code> stylesheet transforms the datapage into the XSL-FO Formatting Objects which, combined with the built-in FOP serializer, render the results in Adobe PDF format.</p>

Setting Up the Demo Data

To set up the demo data do the following:

1. Change directory to the `./demo` directory on your machine.
2. In this directory, run SQLPLUS. Connect to your database as CTXSYS/CTXSYS — the schema owner for Oracle Text (Intermedia Text) packages — and issue the command

```
GRANT EXECUTE ON CTX_DDL TO SCOTT;
```

3. Connect to your database as SYSTEM/MANAGER and issue the command:

```
GRANT QUERY REWRITE TO SCOTT;
```

This allows SCOTT to create a functional index that one of the demos uses to perform case-insensitive queries on descriptions of airports.

4. Connect to your database as SCOTT/TIGER.
5. Run the script `install.sql` in the `./demo` directory. This script runs all SQL scripts for all the demos.

```
install.sql
@@insclaim/insclaim.sql
@@document/docdemo.sql
@@classerr/invalidclasses.sql
@@airport/airport.sql
@@insertxml/newsstory.sql
@@empdept/empdeptobjs.sql
```

6. Change directory to `./doyouxml` subdirectory, and run the following:

```
imp scott/tiger file=doyouxml.dmp
```

to import sample data for the "Do You XML? Site" demo.

7. To experience the Scalable Vector Graphics (SVG) demonstration, install an SVG plug-in into your browser, such as Adobe SVG Plug-in.

Advanced XSQL Pages Topics

Understanding Client Stylesheet-Override Options

If the current XSQL page being requested allows it, you can supply an XSLT stylesheet URL in the request to override the default stylesheet that would have been used — or to apply a stylesheet where none would have been applied by default. The client-initiated stylesheet URL is provided by supplying the `xml-stylesheet` parameter as part of the request. The valid values for this parameter are:

- Any relative URL, interpreted relative to the XSQL page being processed
- Any absolute URL using the `http` protocol scheme, provided it references a trusted host (as defined in the `XSQLConfig.xml` file)
- The literal value `none`

This last value, `xml-stylesheet=none`, is particularly useful during development to temporarily "short-circuit" the XSLT stylesheet processing to see

what XML datagram your stylesheet is actually seeing. This can help understand why a stylesheet might not be producing the expected results.

Client-override of stylesheets for an XSQL page can be disallowed either by:

- Setting the `allow-client-style` configuration parameter to `no` in the `XSQLConfig.xml` file, or
- Explicitly including an `allow-client-style="no"` attribute on the document element of any XSQL page

If client-override of stylesheets has been globally disabled by default in the `XSQLConfig.xml` configuration file, any page can still enable client-override explicitly by including an `allow-client-style="yes"` attribute on the document element of that page.

Controlling How Stylesheets are Processed

Controlling the Content Type of the Returned Document

Setting the content type of the information you serve is very important. It allows the requesting client to correctly interpret the information that you send back. If your stylesheet uses an `<xsl:output>` element, the XSQL Page Processor infers the media type and encoding of the returned document from the `media-type` and `encoding` attributes of `<xsl:output>`.

For example, the following stylesheet uses the `media-type="application/vnd.ms-excel"` attribute on `<xsl:output>` to transform the results of an XSQL page containing a standard query over the `emp` table into Microsoft Excel spreadsheet format.

```
<?xml version="1.0"?>
<!-- empToExcel.xsl -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" media-type="application/vnd.ms-excel"/>
  <xsl:template match="/">
    <html>
      <table>
        <tr><th>EMPNO</th><th>ENAME</th><th>SAL</th></tr>
        <xsl:for-each select="ROWSET/ROW">
          <tr>
            <td><xsl:value-of select="EMPNO"/></td>
            <td><xsl:value-of select="ENAME"/></td>
            <td><xsl:value-of select="SAL"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </html>
  </template>
</xsl:stylesheet>
```



```

        </xsl:for-each>
    </table>
</html>
</xsl:template>
</xsl:stylesheet>

```

An XSQL page that makes use of this stylesheet looks like this:

```

<?xml version="1.0"?>
<?xml-stylesheet href="empToExcel.xsl" type="text/xsl"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
    select * from emp order by sal desc
</xsql:query>

```

Assigning the Stylesheet Dynamically

As we've seen, if you include an `<?xml-stylesheet?>` processing instruction at the top of your `.xsql` file, it will be considered by the XSQL page processor for use in transforming the resulting XML datagram. For example:

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
    <xsql:query>
        SELECT * FROM emp ORDER BY sal DESC
    </xsql:query>
</page>

```

would use the `emp.xsl` stylesheet to transform the results of the EMP query in the server tier, before returning the response to the requestor. The stylesheet is accessed by the relative or absolute URL provided in the `href` pseudo-attribute on the `<?xml-stylesheet?>` processing instruction.

By including one or more parameter references in the value of the `href` pseudo-attribute, you can dynamically determine the name of the stylesheet. For example, this page selects the name of the stylesheet to use from a table by assigning the value of a page-private parameter using a query.

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="{@sheet}.xsl"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
    <xsql:set-page-param bind-params="UserCookie" name="sheet">
        SELECT stylesheet_name
           FROM user_prefs
          WHERE username = ?
    </xsql:set-page-param>

```

```
<xsql:query>
  SELECT * FROM emp ORDER BY sal DESC
</xsql:query>
</page>
```

Processing Stylesheets in the Client

Some browsers like Microsoft's Internet Explorer 5.0 and higher support processing XSLT stylesheets in the client. These browsers recognize the stylesheet to be processed for an XML document in the same way that a server-side XSQL page does, using an `<?xml-stylesheet?>` processing instruction. This is not a coincidence. The use of `<?xml-stylesheet?>` for this purpose is part of the W3C Recommendation from June 29, 1999 entitled "Associating Stylesheets with XML Documents, Version 1.0"

By default, the XSQL page processor performs XSLT transformations in the server, however by adding on additional pseudo-attribute to your `<?xml-stylesheet?>` processing instruction in your XSQL page — `client="yes"` — the page processor will defer the XSLT processing to the client by serving the XML datagram "raw", with the current `<?xml-stylesheet?>` at the top of the document.

One important point to note is that Internet Explorer 5.0 shipped in late 1998, containing an implementation of the XSL stylesheet language that conformed to a December 1998 Working Draft of the standard. The XSLT 1.0 Recommendation that finally emerged in November of 1999 had significant changes from the earlier working draft version on which IE5 is based. This means that IE5 browsers understand a different "dialect" of XSLT than all other XSLT processors — like the Oracle XSLT processor — which implement the XSLT 1.0 Recommendation syntax.

Toward the end of 2000, Microsoft released version 3.0 of their MSXML components as a Web-downloadable release. This latest version *does* implement the XSLT 1.0 standard, however in order for it to be used as the XSLT processor inside the IE5 browser, the user must go through additional installation steps. Unfortunately there is no way for a server to detect that the IE5 browser has installed the latest XSLT components, so until the Internet Explorer 6.0 release emerges — which will contain the latest components by default and which will send a detectably different User-Agent string containing the 6.0 version number — stylesheets delivered for client processing to IE5 browsers should use the earlier IE5-"flavor" of XSL.

What we need is a way to request that an XSQL page use different stylesheets depending on the User-Agent making the request. Luckily, the XSQL Pages framework makes this easy and we learn how in the next section.

Providing Multiple, UserAgent-Specific Stylesheets

You can include multiple `<?xml-stylesheet?>` processing instructions at the top of an XSQL page and any of them can contain an optional `media` pseudo-attribute. If specified, the `media` pseudo-attribute's value is compared case-insensitively with the value of the HTTP header's User-Agent string. If the value of the `media` pseudo-attribute matches a part of the User-Agent string, then the processor selects the current `<?xml-stylesheet?>` processing instruction for use, otherwise it ignores it and continues looking. The first matching processing instruction in document order will be used. A processing instruction *without* a `media` pseudo-attribute matches all user agents so it can be used as the fallback/default.

For example, the following processing instructions at the top of an `.xsql` file...

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="lynx" href="doyouxml-lynx.xsl" ?>
<?xml-stylesheet type="text/xsl" media="msie 5" href="doyouxml-ie.xsl" ?>
<?xml-stylesheet type="text/xsl" href="doyouxml.xsl" ?>
<page xmlns:xsql="urn:oracle-xsql" connection="demo">
:
```

will use `doyouxml-lynx.xsl` for Lynx browsers, `doyouxml-ie.xsl` for Internet Explorer 5.0 or 5.5 browsers, and `doyouxml.xsl` for all others.

[Table 10-14](#) summarizes all of the supported pseudo-attributes allowed on the `<?xml-stylesheet?>` processing instruction.

Table 10-14 Pseudo-Attributes for `<?xml-stylesheet?>`

Attribute Name	Description
<code>type = "string"</code>	Indicates the MIME type of the associated stylesheet. For XSLT stylesheets, this attribute must be set to the string <code>text/xsl</code> . This attribute may be present <i>or</i> absent when using the <code>serializer</code> attribute, depending on whether an XSLT stylesheet should execute before invoking the serializer or not.
<code>href = "URL"</code>	Indicates the relative or absolute URL to the XSLT stylesheet to be used. If an absolute URL is supplied that uses the <code>http</code> protocol scheme, the IP address of the resource must be a trusted host listed in the <code>XSQLConfig.xml</code> file.
<code>media = "string"</code>	This attribute is optional. If provided, its value is used to perform a case- <i>insensitive</i> match on the User-Agent string from the HTTP header sent by the requesting device. The current <code><?xml-stylesheet?></code> processing instruction will only be used if the User-Agent string contains the value of the <code>media</code> attribute, otherwise it is ignored.

Table 10–14 Pseudo-Attributes for <?xml-stylesheet?>

Attribute Name	Description
<code>client = "boolean"</code>	If set to <code>yes</code> , caused the XSQL page processor to defer the processing of the associated XSLT stylesheet to the client. The "raw" XML datagram will be sent to the client with the current <code><?xml-stylesheet?></code> processing instruction at the top of the document. The default if not specified is to perform the transform in the server.
<code>serializer = "string"</code>	<p>By default, the XSQL page processor uses the:</p> <ul style="list-style-type: none"> ■ XML DOM serializer if no XSLT stylesheet is used ■ XSLT processor's serializer, if XSLT stylesheet is used <p>Specifying this pseudo-attribute indicates that a custom serializer implementation should be used instead.</p> <p>Valid values are <i>either</i> the name of a custom serializer defined in the <code><serializerdefs></code> section of the <code>XSQLConfig.xml</code> file, or the string <code>java:fully.qualified.Classname</code>. If both an XSLT stylesheet and the serializer attribute are present, then the XSLT transform is performed first, then the custom serializer is invoked to render the final result to the <code>OutputStream</code> or <code>PrintWriter</code>.</p>

Using XSQLConfig.xml to Tune Your Environment

Use the `XSQLConfig.xml` File to tune your XSQL pages environment. [Table 10–15](#) defines all of the parameters that can be set.

Table 10–15 XSQLConfig.xml Configuration Settings

Configuration Setting Name
<code>XSQLConfig/servlet/output-buffer-size</code>
Sets the size (in bytes) of the buffered output stream. If your servlet engine already buffers I/O to the Servlet Output Stream, then you can set to 0 to avoid additional buffering.
Default value is 0. Valid value is any non-negative integer.

Table 10–15 XSQLConfig.xml Configuration Settings**Configuration Setting Name****XSQLConfig/servlet/suppress-mime-charset/media-type**

The XSQL Servlet sets the HTTP `ContentType` header to indicate the MIME type of the resource being returned to the request. By default, the XSQL Servlet includes the optional character set information in the MIME type. For a particular MIME type, you can suppress the inclusion of the character set information by including a `<media-type>` element, with the desired MIME type as its contents.

You may list any number of `<media-type>` elements.

Valid value is any string.

XSQLConfig/processor/character-set-conversion/default-charset

By default, the XSQL page processor does character set conversion on the value of HTTP parameters to compensate for the default character set used by most servlet engines. The default base character set used for conversion is the Java character set `8859_1` corresponding to IANA's `ISO-8859-1` character set. If your servlet engine uses a different character set as its base character set you can now specify that value here.

To suppress character set conversion, specify the empty element `<none/>` as the content of the `<default-charset>` element, instead of a character set name. This is useful if you are working with parameter values that are correctly representable using your servlet's default character set, and eliminates a small amount of overhead associated with performing the character set conversion.

Valid values are any Java character set name, or the element `<none/>`.

XSQLConfig/processor/reload-connections-on-error

Connection definitions are cached when the XSQL Page Processor is initialized. Set this setting to `yes` to cause the processor to reread the `XSQLConfig.xml` file to reload connection definitions if an attempt is made to request a connection name that's not in the cached connection list. The `yes` setting is useful during development when you might be adding new `<connection>` definitions to the file while the servlet is running. Set to `no` to avoid reloading the connection definition file when a connection name is not found in the in-memory cache.

Default is `yes`. Valid values are `yes` and `no`.

XSQLConfig/processor/default-fetch-size

Sets the default value of the row fetch size for retrieving information from SQL queries from the database. Only takes effect if you are using the Oracle JDBC Driver, otherwise the setting is ignored. Useful for reducing network roundtrips to the database from the servlet engine running in a different tier.

Default is 50. Valid value is any non-zero positive integer.

Table 10–15 XSQLConfig.xml Configuration Settings

Configuration Setting Name
XSQLConfig/processor/page-cache-size Sets the size of the XSQL cache for XSQL page templates. This determines the maximum number of XSQL pages that will be cached. Least recently used pages get "bumped" out of the cache if you go beyond this number. Default is 25. Valid value is any non-zero positive integer.
XSQLConfig/processor/stylesheet-cache-size Sets the size of the XSQL cache for XSLT stylesheets. This determines the maximum number of stylesheets that will be cached. Least recently used stylesheets get "bumped" out of the cache if you go beyond this number. Default is 25. Valid value is any non-zero positive integer.
XSQLConfig/processor/stylesheet-pool/initial Each cached stylesheet is actually a pool of cached stylesheet instances to improve throughput. Sets the initial number of stylesheets to be allocated in each stylesheet pool. Default is 1. Valid value is any non-zero positive integer.
XSQLConfig/processor/stylesheet-pool/increment Sets the number of stylesheets to be allocated when the stylesheet pool must grow due to increased load on the server. Default is 1. Valid value is any non-zero positive integer.
XSQLConfig/processor/stylesheet-pool/timeout-seconds Sets the number of seconds of inactivity that must transpire before a stylesheet instance in the pool will be removed to free resources as the pool tries to "shrink" back to its initial size. Default is 60. Valid value is any non-zero positive integer.
XSQLConfig/processor/connection-pool/initial The XSQL page processor's default connection manager implements connection pooling to improve throughput. This setting controls the initial number of JDBC connections to be allocated in each connection pool. Default is 2. Valid value is any non-zero positive integer.
XSQLConfig/processor/connection-pool/increment Sets the number of connections to be allocated when the connection pool must grow due to increased load on the server. Default is 1. Valid value is any non-zero positive integer.

Table 10–15 XSQLConfig.xml Configuration Settings**Configuration Setting Name****XSQLConfig/processor/connection-pool/timeout-seconds**

Sets the number of seconds of inactivity that must transpire before a JDBC connection in the pool will be removed to free resources as the pool tries to "shrink" back to its initial size.

Default is 60. Valid value is any non-zero positive integer.

XSQLConfig/processor/connection-pool/dump-allowed

Determines whether a diagnostic report of connection pool activity can be requested by passing the `dump-pool=y` parameter in the page request.

Default is no. Valid value is yes or no.

XSQLConfig/processor/connection-manager/factory

Specifies the fully-qualified Java class name of the XSQL connection manager factory implementation. If not specified, this setting defaults to `oracle.xml.xsql.XSQLConnectionFactoryImpl`.

Default is `oracle.xml.xsql.XSQLConnectionFactoryImpl`. Valid value is any class name that implements the `oracle.xml.xsql.XSQLConnectionFactory` interface.

XSQLConfig/processor/timing/page

Determines whether a the XSQL page processor adds an `xsql-timing` attribute to the document element of the page whose value reports the elapsed number of milliseconds required to process the page.

Default is no. Valid value is yes or no.

XSQLConfig/processor/timing/action

Determines whether a the XSQL page processor adds comment to the page just before the action element whose contents reports the elapsed number of milliseconds required to process the action.

Default is no. Valid value is yes or no.

Table 10–15 XSQLConfig.xml Configuration Settings

Configuration Setting Name

XSQLConfig/processor/security/stylesheet/defaults/allow-client-style

While developing an application, it is frequently useful to take advantage of the XSQL page processor's per-request stylesheet override capability by providing a value for the special `xml-stylesheet` parameter in the request. One of the most common uses is to provide the `xml-stylesheet=none` combination to temporarily disable the application of the stylesheet to "peek" underneath at the raw XSQL data page for debugging purposes.

When development is completed, you could explicitly add the `allow-client-style="no"` attribute to the document element of each XSQL page to prohibit client overriding of the stylesheet in the production application. However, using this configuration setting, you can globally change the default behavior for `allow-client-style` in a single place.

Note that this only provides the *default* setting for this behavior. If the `allow-client-style="yes|no"` attribute is explicitly specified on the document element for a given XSQL page, its value takes precedence over this global default.

Valid values are `yes` and `no`.

XSQLConfig/processor/security/stylesheet/trusted-hosts/host

XSLT stylesheets can invoke extension functions. In particular, the Oracle XSLT processor — which the XSQL page processor uses to process all XSLT stylesheets — supports *Java* extension functions. Typically your XSQL pages will refer to XSLT stylesheets using relative URL's. The XSQL page processor enforces that any absolute URL to an XSLT stylesheet that is processed must be from a trusted host whose name is listed here in the configuration file.

You may list any number of `<host>` elements inside the `<trusted-hosts>` element. The name of the local machine, `localhost`, and `127.0.0.1` are considered trusted hosts by default.

Valid values are any hostname or IP address.

XSQLConfig/http/proxyhost

Sets the name of the HTTP proxy server to use when processing URL's with the `http` protocol scheme.

Valid value is any hostname or IP address.

XSQLConfig/http/proxyport

Sets the port number of the HTTP proxy server to use when processing URL's with the `http` protocol scheme.

Valid value is any non-zero integer.

Table 10–15 XSQLConfig.xml Configuration Settings

Configuration Setting Name
XSQLConfig/connectiondefs/connection
Defines a "nickname" and the JDBC connection details for a named connection for use by the XSQL page processor.
You may supply any number of <connection> element children of <connectiondefs>. Each connection definition must supply a name attribute, and may supply appropriate children elements <username>, <password>, <driver>, <dburl>, and <autocommit>.
XSQLConfig/connectiondefs/connection/username
Defines the username for the current connection.
XSQLConfig/connectiondefs/connection/password
Defines the password for the current connection.
XSQLConfig/connectiondefs/connection/dburl
Defines the JDBC connection URL for the current connection.
XSQLConfig/connectiondefs/connection/driver
Specifies the fully-qualified Java class name of the JDBC driver to be used for the current connection. If not specified, defaults to <code>oracle.jdbc.driver.OracleDriver</code> .
XSQLConfig/connectiondefs/connection/autocommit
Explicitly sets the Auto Commit flag for the current connection. If not specified, connection uses JDBC driver's default setting for Auto Commit.
XSQLConfig/serializerdefs/serializer
Defines a named custom serializer implementation.
You may supply any number of <serializer> element children of <serializerdefs>. Each must specify both a <name> and a <class> child element.
XSQLConfig/serializerdefs/serializer/name
Defines the name of the current custom serializer definition.
XSQLConfig/connectiondefs/connection/class
Specifies the fully-qualified Java class name of the current custom serializer. The class must implement the <code>oracle.xml.xsql.XSQLDocumentSerializer</code> interface.

Using the FOP Serializer to Produce PDF Output

Using the XSQL Pages framework's support for custom serializers, the `oracle.xml.xsql.serializers.XSQLFOPSerializer` is provided for integrating with the Apache FOP processor (<http://xml.apache.org/fop>). The FOP

processor renders a PDF document from an XML document containing XSL Formatting Objects (<http://www.w3.org/TR/xsl>).

For example, given the following XSLT stylesheet, `EmpTableFO.xsl`:

```
<!-- EmpTableFO.xsl -->
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xsl:version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- defines the layout master -->
  <fo:layout-master-set>
    <fo:simple-page-master master-name="first"
      page-height="29.7cm"
      page-width="21cm"
      margin-top="1cm"
      margin-bottom="2cm"
      margin-left="2.5cm"
      margin-right="2.5cm">
      <fo:region-body margin-top="3cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <!-- starts actual layout -->
  <fo:page-sequence master-name="first">
    <fo:flow flow-name="xsl-region-body">
      <fo:block font-size="24pt" font-family="Garamond" line-height="24pt"
        space-after.optimum="3pt" font-weight="bold"
        start-indent="15pt">
        Total of All Salaries is $<xsl:value-of select="sum(/ROWSET/ROW/SAL)"/>
      </fo:block>
      <!-- Here starts the table -->
      <fo:block border-width="2pt">
        <fo:table>
          <fo:table-column column-width="4cm"/>
          <fo:table-column column-width="4cm"/>
          <fo:table-body font-size="10pt" font-family="sans-serif">
            <xsl:for-each select="ROWSET/ROW">
              <fo:table-row line-height="12pt">
                <fo:table-cell>
                  <fo:block><xsl:value-of select="ENAME"/></fo:block>
                </fo:table-cell>
                <fo:table-cell>
                  <fo:block><xsl:value-of select="SAL"/></fo:block>
                </fo:table-cell>
              </fo:table-row>
            </xsl:for-each>
          </fo:table-body>
        </fo:table>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

```

    </fo:block>
  </fo:flow>
</fo:page-sequence>
</fo:root>

```

you can format the results of a query against the EMP table using the supplied FOP serializer (pre-defined in `XSQLConfig.xml`) with an XSQL page like:

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emptablefo.xsl" serializer="FOP"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT ENAME, SAL FROM EMP
  ORDER BY SAL asc
</xsql:query>

```

Note: To use the XSQL FOP Serializer, you need to add these additional Java archives to your server's CLASSPATH:

- `xsqlserializers.jar` — **supplied with Oracle XSQL**
 - `fop.jar` — **From Apache, version 0.16 or higher**
 - `w3c.jar` — **from the FOP distribution's `./lib` directory**
-
-

Using XSQL Page Processor Programmatically

The `XSQLRequest` class, allows you to utilize the XSQL page processor "engine" from within your own custom Java programs. Using the API is simple. You construct an instance of `XSQLRequest`, passing the XSQL page to be processed into the constructor as one of the following:

- String containing a URL to the page
- URL object for the page
- In-memory `XMLDocument`

Then you invoke one of the following methods to process the page:

- `process()` — to write the result to a `PrintWriter` or `OutputStream`, or
- `processToXML()` — to return the result as an XML Document

If you want to use the built-in XSQL Connection Manager — which implements JDBC connection pooling based on `XSQLConfig.xml`-based connection definitions — then the XSQL page is all you need to pass to the constructor. Optionally, you can

pass in a custom implementation for the `XSQLConnectionFactory` interface as well, if you want to use your own connection manager implementation.

Note that the ability to pass the XSQL page to be processed as an in-memory XML Document object means that you can dynamically generate any valid XSQL page for processing using any means necessary, then pass the page to the XSQL engine for evaluation.

When processing a page, there are two additional things you may want to do as part of the request:

- Pass a set of parameters to the request
You accomplish this by passing any object that implements the `Dictionary` interface, to the `process()` or `processToXML()` methods. Passing a `HashTable` containing the parameters is one popular approach.
- Set an XML document to be processed by the page as if it were the "posted XML" message body
You can do this using the `setPostedDocument()` method on the `XSQLRequest` object.

Here is a simple example of processing a page using `XSQLRequest`:

```
import oracle.xml.xsql.XSQLRequest;
import java.util.Hashtable;
import java.io.PrintWriter;
import java.net.URL;
public class XSQLRequestSample {
    public static void main( String[] args) throws Exception {
        // Construct the URL of the XSQL Page
        URL pageUrl = new URL("file:///C:/foo/bar.xsql");
        // Construct a new XSQL Page request
        XSQLRequest req = new XSQLRequest(pageUrl);
        // Setup a Hashtable of named parameters to pass to the request
        Hashtable params = new Hashtable(3);
        params.put("param1", "value1");
        params.put("param2", "value2");
        /* If needed, treat an existing, in-memory XMLDocument as if
        ** it were posted to the XSQL Page as part of the request
        req.setPostedDocument(myXMLDocument);
        **
        */
        // Process the page, passing the parameters and writing the output
        // to standard out.
        req.process(params, new PrintWriter(System.out))
    }
}
```

```

        ,new PrintWriter(System.err));
    }
}

```

Writing Custom XSQL Action Handlers

When the task at hand requires custom processing, and none of the built-in actions does exactly what you need, you can augment your repertoire by writing your own actions that any of your XSQL pages can use.

The XSQL page processor at its very core is an engine that processes XML documents containing "action elements". The page processor engine is written to support any action that implements the `XSQLActionHandler` interface. All of the built-in actions implement this interface.

The XSQL Page Processor processes the actions in a page in the following way. For each action in the page, the engine:

1. Constructs an instance of the action handler class using the default constructor
2. Initializes the handler instance with the action element object and the page processor context by invoking the method:

```
init(Element actionElt,XSQLPageRequest context)
```

3. Invokes the method that allows the handler to handle the action:

```
handleAction (Node result)
```

For built-in actions, the engine knows the mapping of XSQL action element name to the Java class that implements the action's handler. [Table 10-16](#) lists that mapping explicitly for your reference. For user-defined actions, you use the built-in:

```
<xsql:action handler="fully.qualified.Classname" ... />
```

action whose `handler` attribute provides the fully-qualified name of the Java class that implements the custom action handler.

Table 10-16 Built-In XSQL Elements and Action Handler Classes

XSQL Action Element	Handler Class in <code>oracle.xml.xsql.actions</code>
<code><xsql:query></code>	<code>XSQLQueryHandler</code>
<code><xsql:dml></code>	<code>XSQLDMLHandler</code>
<code><xsql:set-stylesheet-param></code>	<code>XSQLStylesheetParameterHandler</code>
<code><xsql:insert-request></code>	<code>XSQLInsertRequestHandler</code>

Table 10–16 Built-In XSQL Elements and Action Handler Classes

XSQL Action Element	Handler Class in oracle.xml.xsql.actions
<code><xsql:include-xml></code>	<code>XSQLIncludeXMLHandler</code>
<code><xsql:include-request-params></code>	<code>XSQLIncludeRequestHandler</code>
<code><xsql:include-xsql></code>	<code>XSQLIncludeXSQLHandler</code>
<code><xsql:include-owa></code>	<code>XSQLIncludeOWAHandler</code>
<code><xsql:action></code>	<code>XSQLExtensionActionHandler</code>
<code><xsql:ref-cursor-function></code>	<code>XSQLRefCursorFunctionHandler</code>
<code><xsql:include-param></code>	<code>XSQLGetParameterHandler</code>
<code><xsql:set-session-param></code>	<code>XSQLSetSessionParamHandler</code>
<code><xsql:set-page-param></code>	<code>XSQLSetPageParamHandler</code>
<code><xsql:set-cookie></code>	<code>XSQLSetCookieHandler</code>
<code><xsql:insert-param></code>	<code>XSQLInsertParameterHandler</code>
<code><xsql:update-request></code>	<code>XSQLUpdateRequestHandler</code>
<code><xsql:delete-request></code>	<code>XSQLDeleteRequestHandler</code>

Writing your Own Action Handler

To create a custom Action Handler, you need to provide a class that implements the `oracle.xml.xsql.XSQLActionHandler` interface. Most custom action handlers should extend `oracle.xml.xsql.XSQLActionHandlerImpl` that provides a default implementation of the `init()` method and offers a set of useful helper methods that will prove very useful.

When an action handler's `handleAction` method is invoked by the XSQL page processor, the action implementation gets passed the root node of a DOM Document Fragment to which the action handler should append any dynamically created XML content that should be returned to the page.

The XSQL Page Processor conceptually replaces the action element in the XSQL page *template* with the content of this Document Fragment. It is completely legal for an Action Handler to append *nothing* to this document fragment, if it has no XML content to add to the page.

While writing you custom action handlers, several methods on the `XSQLActionHandlerImpl` class are worth noting because they make your life a lot easier. [Table 10–17](#) lists the methods that will likely come in handy for you.

Table 10–17 Helpful Methods on `oracle.xml.xsql.SQLActionHandlerImpl`

Method Name	Description
<code>getActionElement</code>	Returns the current action element being handled
<code>getActionElementContent</code>	Returns the text content of the current action element, with all lexical parameters substituted appropriately.
<code>getPageRequest</code>	<p>Returns the current XSQL page processor context. Using this object you can then do things like:</p> <ul style="list-style-type: none"> ■ <code>setPageParam()</code> Set a page parameter value ■ <code>getPostedDocument()/setPostedDocument()</code> Get or set the posted XML document ■ <code>translateURL()</code> Translate a relative URL to an absolute URL ■ <code>getRequestObject()/setRequestObject()</code> Get or set objects in the page request context that can be shared across actions in a single page. ■ <code>getJDBCConnection()</code> Gets the JDBC connection in use by this page (possible null if no connection in use). ■ <code>getRequestType()</code> Detect whether you are running in the "Servlet", "Command Line" or "Programmatic" context. For example, if the request type is "Servlet" then you can cast the <code>XSQLPageRequest</code> object to the more specific <code>XSQLServletPageRequest</code> to access addition Servlet-specific methods like <code>getHttpServletRequest</code>, <code>getHttpServletResponse</code>, and <code>getServletContext</code>
<code>getAttributeAllowingParam</code>	Retrieve the attribute value from an element, resolving any XSQL lexical parameter references that might appear in the attribute's value. Typically this method is applied to the action element itself, but it is also useful for accessing attributes of any of its subelements. To access an attribute value without allowing lexical parameters, use the standard <code>getAttribute()</code> method on the DOM Element interface.

Table 10–17 Helpful Methods on `oracle.xml.xsql.SQLActionHandlerImpl`

Method Name	Description
<code>appendSecondaryDocument</code>	Append the entire contents of an external XML document to the root of the action handler result content.
<code>addResultElement</code>	Simplify appending a single element with text content to the root of the action handler result content.
<code>firstColumnOfFirstRow</code>	Return the first column value of the first row of a SQL statement passed in. Requires the current page to have a connection attribute on its document element, or an error is returned.
<code>bindVariableCount</code>	Returns the number of tokens in the space-separated list of <code>bind-params</code> , indicating how many bind variables are expected to be bound to parameters.
<code>handleBindVariables</code>	Manage the binding of JDBC bind variables that appear in a prepared statement with the parameter values specified in the <code>bind-params</code> attribute on the current action element. If the statement already is using a number of bind variables prior to call this method, you can pass the number of existing bind variable "slots" in use as well.
<code>reportErrorIncludingStatement</code>	Report an error, including the offending (SQL) statement that caused the problem, optionally including a numeric error code.
<code>reportFatalError</code>	Report a fatal error.
<code>reportMissingAttribute</code>	Report an error that a required action handler attribute is missing using the standard <code><xsql-error></code> element.
<code>reportStatus</code>	Report action handler status using the standard <code><xsql-status></code> element.
<code>requiredConnectionProvided</code>	Checks whether a connection is available for this request, and outputs an "errorgram" into the page if no connection is available.
<code>variableValue</code>	Returns the value of a lexical parameter, taking into account all scoping rules which might determine its default value.

The following example shows a custom action handler `MyIncludeXSQLHandler` that leverages one of the built-in action handlers and then uses arbitrary Java code to modify the resulting XML fragment returned by that handler before appending its result to the XSQL page:


```

import oracle.xml.xsql.*;
import oracle.xml.xsql.actions.XSQLIncludeXSQLHandler;
import org.w3c.dom.*;
import java.sql.SQLException;
public class MyIncludeXSQLHandler extends XSQLActionHandlerImpl {
    XSQLActionHandler nestedHandler = null;
    public void init(XSQLPageRequest req, Element action) {
        super.init(req, action);
        // Create an instance of an XSQLIncludeXSQLHandler
        // and init() the handler by passing the current request/action
        // This assumes the XSQLIncludeXSQLHandler will pick up its
        // href="xxx.xsql" attribute from the current action element.
        nestedHandler = new XSQLIncludeXSQLHandler();
        nestedHandler.init(req,action);
    }
    public void handleAction(Node result) throws SQLException {
        DocumentFragment df=result.getOwnerDocument().createDocumentFragment();
        nestedHandler.handleAction(df);
        // Custom Java code here can work on the returned document fragment
        // before appending the final, modified document to the result node.
        // For example, add an attribute to the first child
        Element e = (Element)df.getFirstChild();
        if (e != null) {
            e.setAttribute("ExtraAttribute", "SomeValue");
        }
        result.appendChild(df);
    }
}

```

If you create custom action handlers that need to work differently based on whether the page is being requested through the XSQL Servlet, the XSQL Command Line Utility, or programmatically through the XSQLRequest class, then in your Action Handler implementation you can call `getPageRequest()` to get a reference to the XSQLPageRequest interface for the current page request. By calling `getRequestType()` on the XSQLPageRequest object, you can see if the request is coming from the “Servlet”, “Command Line”, or “Programmatic” routes respectively. If the return value is “Servlet”, then you can get access to the HTTP Servlet's request, response, and servlet context objects by doing:

```

XSQLServletPageRequest xspr = (XSQLServletPageRequest)getPageRequest();
if (xspr.getRequestType().equals("Servlet")) {
    HttpServletRequest req = xspr.getHttpServletRequest();
    HttpServletResponse resp = xspr.getHttpServletResponse();
    ServletContext cont = xspr.getServletContext();
    // do something fun here with req, resp, or cont however
}

```

```
// writing to the response directly from a handler will
// produce unexpected results. Allow the XSQL Servlet
// or your custom Serializer to write to the servlet's
// response output stream at the write moment later when all
// action elements have been processed.
}
```

Writing Custom XSQL Serializers

You can provide a user-defined serializer class to programmatically control how the final XSQL datapage's XML document should be serialized to a text or binary stream. A user-defined serializer must implement the `oracle.xml.xsql.XSQLDocumentSerializer` interface which comprises the single method:

```
void serialize(org.w3c.dom.Document doc, XSQLPageRequest env) throws Throwable;
```

In this release, DOM-based serializers are supported. A future release may support SAX2-based serializers as well. A custom serializer class is expected to perform the following tasks in the correct order:

1. Set the content type of the serialized stream before writing any content to the output `PrintWriter` (or `OutputStream`).

You set the type by calling `setContentType()` on the `XSQLPageRequest` that is passed to your serializer. When setting the content type, you can either set just a MIME type like this:

```
env.setContentType("text/html");
```

or a MIME type with an explicit output encoding character set like this:

```
env.setContentType("text/html;charset=Shift_JIS");
```

2. Call `getWriter()` or `getOutputStream()` — but not both! — on the `XSQLPageRequest` to get the appropriate `PrintWriter` or `OutputStream` respectively to use for serializing the content.

For example, the following custom serializer illustrates a simple implementation which simply serializes an HTML document containing the name of the document element of the current XSQL data page:

```
package oracle.xml.xsql.serializers;
import org.w3c.dom.Document;
import java.io.PrintWriter;
import oracle.xml.xsql.*;
```

```

public class XSQLSampleSerializer implements XSQLDocumentSerializer {
    public void serialize(Document doc, XSQLPageRequest env) throws Throwable {
        String encoding = env.getPageEncoding(); // Use same encoding as XSQL page
                                                // template. Set to specific
                                                // encoding if necessary

        String mimeType = "text/html"; // Set this to the appropriate content type
        // (1) Set content type using the setContentType on the XSQLPageRequest
        if (encoding != null && !encoding.equals("")) {
            env.setContentType(mimeType+"; charset="+encoding);
        }
        else {
            env.setContentType(mimeType);
        }
        // (2) Get the output writer from the XSQLPageRequest
        PrintWriter e = env.getWriter();
        // (3) Serialize the document to the writer
        e.println("<html>Document element is <b>"+
            doc.getDocumentElement().getNodeName()+
            "</b></html>");
    }
}

```

There are two ways to use a custom serializer, depending on whether you need to first perform an XSLT transformation before serializing or not. To perform an XSLT transformation before using a custom serializer, simply add the `serializer="java:fully.qualified.ClassName"` in the `<?xml-stylesheet?>` processing instruction at the top of your page like this:

```

<?xml version="1.0?>
<?xml-stylesheet type="text/xsl" href="mystyle.xsl"
    serializer="java:my.pkg.MySerializer"?>

```

If you only need the custom serializer, simply leave out the `type` and `href` attributes like this:

```

<?xml version="1.0?>
<?xml-stylesheet serializer="java:my.pkg.MySerializer"?>

```

You can also assign a short nickname to your custom serializers in the `<serializerdefs>` section of the `XSQLConfig.xml` file and then use the nickname (case-sensitive) in the `serializer` attribute instead to save typing. For example, if you have the following in `XSQLConfig.xml`:

```
<XSQLConfig>
  <!-- etc. -->
  <serializerdefs>
    <serializer>
      <name>Sample</name>
      <class>oracle.xml.xsql.serializers.XSQLSampleSerializer</class>
    </serializer>
    <serializer>
      <name>FOP</name>
      <class>oracle.xml.xsql.serializers.XSQLFOPSerializer</class>
    </serializer>
  </serializerdefs>
</XSQLConfig>
```

then you can use the nicknames "Sample" and/or "FOP" as shown in the following examples:

```
<?xml-stYLESHEET type="text/xsl" href="emp-to-xslfo.xsl" serializer="FOP"?>
```

or

```
<?xml-stYLESHEET serializer="Sample"?>
```

The `XSQLPageRequest` interface supports both a `getWriter()` and a `getOutputStream()` method. Custom serializers can call `getOutputStream()` to return an `OutputStream` instance into which binary data (like a dynamically produced GIF image, for example) can be serialized. Using the XSQL Servlet, writing to this output stream results in writing the binary information to the servlet's output stream.

For example, the following serializer illustrates an example of writing out a dynamic GIF image. In this example the GIF image is a static little "ok" icon, but it shows the basic technique that a more sophisticated image serializer would need to use:

```
package oracle.xml.xsql.serializers;
import org.w3c.dom.Document;
import java.io.*;
import oracle.xml.xsql.*;

public class XSQLSampleImageSerializer implements XSQLDocumentSerializer {
    // Byte array representing a small "ok" GIF image
    private static byte[] okGif =
        {(byte)0x47, (byte)0x49, (byte)0x46, (byte)0x38,
         (byte)0x39, (byte)0x61, (byte)0xB, (byte)0x0,
         (byte)0x9, (byte)0x0, (byte)0xFFFFFFFF80, (byte)0x0,
```

```
(byte)0x0, (byte)0x0, (byte)0x0, (byte)0x0,
(byte)0xFFFFFFFF, (byte)0xFFFFFFFF, (byte)0xFFFFFFFF, (byte)0x2C,
(byte)0x0, (byte)0x0, (byte)0x0, (byte)0x0,
(byte)0xB, (byte)0x0, (byte)0x9, (byte)0x0,
(byte)0x0, (byte)0x2, (byte)0x14, (byte)0FFFFFFF8C,
(byte)0xF, (byte)0FFFFFFFA7, (byte)0FFFFFFFB8, (byte)0FFFFFFF9B,
(byte)0xA, (byte)0FFFFFFFA2, (byte)0x79, (byte)0FFFFFFFE9,
(byte)0FFFFFFF85, (byte)0x7A, (byte)0x27, (byte)0FFFFFFF93,
(byte)0x5A, (byte)0FFFFFFFE3, (byte)0FFFFFFFEC, (byte)0x75,
(byte)0x11, (byte)0FFFFFFF85, (byte)0x14, (byte)0x0,
(byte)0x3B};
```

```
public void serialize(Document doc, XSQLPageRequest env) throws Throwable {
    env.setContentType("image/gif");
    OutputStream os = env.getOutputStream();
    os.write(okGif, 0, okGif.length);
    os.flush();
}
}
```

Using the XSQL Command Line utility, the binary information is written to the target output file. Using the XSQLRequest programmatic API, two constructors exist that allow the caller to supply the target OutputStream to use for the results of page processing.

Note that your serializer must either call `getWriter()` (for textual output) or `getOutputStream()` (for binary output) but not both. Calling both in the same request will raise an error.

Writing Custom XSQL Connection Managers

You can provide a custom connection manager to replace the built-in connection management mechanism. To provide a custom connection manager implementation, you must provide:

1. A connection manager *factory* object that implements the `oracle.xml.xsql.XSQLConnectionFactory` interface.
2. A connection manager object that implements the `oracle.xml.xsql.XSQLConnectionManager` interface.

Your custom connection manager factory can be set to be used as the default connection manager factory by providing the classname in the `XSQLConfig.xml` file in the section:

```
<!--
| Set the name of the XSQL Connection Manager Factory
| implementation. The class must implement the
| oracle.xml.xsql.XSQLConnectionFactory interface.
| If unset, the default is to use the built-in connection
| manager implementation in
| oracle.xml.xsql.XSQLConnectionFactoryImpl
+-->
<connection-manager>
  <factory>oracle.xml.xsql.XSQLConnectionFactoryImpl</factory>
</connection-manager>
```

In addition to specifying the default connection manager factory, a custom connection factory can be associated with any individual `XSQLRequest` object using API's provided.

The responsibility of the `XSQLConnectionFactory` is to return an instance of an `XSQLConnectionManager` for use by the current request. In a multithreaded environment like a servlet engine, it is the responsibility of the `XSQLConnectionManager` object to insure that a single `XSQLConnection` instance is not used by two different threads. This can be assured by marking the connection as "in use" for the span of time between the invocation of the `getConnection()` method and the `releaseConnection()` method. The default XSQL connection manager implementation automatically pools named connections, and adheres to this threadsafe policy.

Formatting XSQL Action Handler Errors

Errors raised by the processing of any XSQL Action Elements are reported as XML elements in a uniform way so that XSL Stylesheets can detect their presence and optionally format them for presentation.

The action element in error will be replaced in the page by:

```
<xsql-error action="xxx">
```

Depending on the error the `<xsql-error>` element contains:

- A nested `<message>` element
- A `<statement>` element with the offending SQL statement

Displaying Error Information on Screen

Here is an example of an XSLT stylesheet that uses this information to display error information on the screen:

```
<xsl:if test="//xsql-error">
  <table style="background:yellow">
    <xsl:for-each select="//xsql-error">
      <tr>
        <td><b>Action</b></td>
        <td><xsl:value-of select="@action" /></td>
      </tr>
      <tr valign="top">
        <td><b>Message</b></td>
        <td><xsl:value-of select="message" /></td>
      </tr>
    </xsl:for-each>
  </table>
</xsl:if>
```

XSQL Servlet Limitations

XSQL Servlet has the following limitations:

HTTP Parameters with Multibyte Names

HTTP parameters with multibyte names, for example, a parameter whose name is in Kanji, are properly handled when they are inserted into your XSQL page using `<xsql:include-request-params>`. An attempt to refer to a parameter with a multibyte name inside the query statement of an `<xsql:query>` tag will return an empty string for the parameter's value.

Workaround

As a workaround use a non-multibyte parameter name. The parameter can still have a multibyte value which can be handled correctly.

CURSOR() Function in SQL Statements

If you use the CURSOR() function in SQL statements you may get an "Exhausted ResultSet" error if the CURSOR() statements are nested and if the first row of the query returns an empty result set for its CURSOR() function.

Frequently Asked Questions (FAQs) - XSQL Servlet

Specifying a DTD While Transforming XSQL Output to a WML Document

I am trying to write my own stylesheet for transforming XSQL output to WML and VML format. These programs (mobile phone simulators) need a WML document with a specific DTD assigned.

Is there any way, I can specify a particular DTD while transforming XSQL's output to a WML document.

Answer

Sure. The way you do it is using a built-in facility of the XSLT stylesheet called `<xsl:output>`.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output type="xml" doctype-system="your.dtd"/>
  <xsl:template match="/">
    </xsl:template>
    :
    :
  </xsl:stylesheet>
```

This will produce an XML result with a:

```
<!DOCTYPE xxxx SYSTEM "your.dtd">
```

in the result. "your.dtd" can be any valid absolute or relative URL.

XSQL Servlet Conditional Statements

Is it possible to write conditional statements in a .xsql file, and if yes what is the syntax to do that?

For example:

```
<xsql:choose>
  <xsql:when test="@security='admin'">
    <xsql:query>
      SELECT ....
    </xsql:query>
  </xsql:when>
  <xsql:when test="@security='user'">
    <xsql:query>
      SELECT ....
    </xsql:query>
  </xsql:when>
</xsql:choose>
```



```
        </xsql:query>
    </xsql:when>
</xsql:if>
```

Answer

Use `<xsql:ref-cursor-function>` to call a PL/SQL procedure that would conditionally return a REF CURSOR to the appropriate query.

Using Value Retrieved in One Query in Another Query's Where Clause

I have two queries in an .xsql file.

```
<xsql:query>
    select col1,col2
    from table1
</xsql:query>
<xsql:query>
    select col3,col4 from table2
    where col3 = {@col1}    => the value of col1 in the previous query
</xsql:query>
```

How can I use, in the second query, the value of a select list item of the first query?

Answer

You do this with page parameters.

```
<page xmlns:xsql="urn:oracle-xsql" connection="demo">
  <!-- Value of page param "xxx" will be first column of first row -->
  <xsql:set-page-param name="xxx">
    select one from table1 where ...
  </xsql:set-page-param>
  <xsql:query bind-params="xxx">
    select col3,col4 from table2
    where col3 = ?
  </xsql:query>
</page>
```

Working with Non-Oracle Databases

Can the XSQL Servlet connect to any DB that has JDBC support?

Answer

Yes. Just indicate the appropriate JDBC driver class and connection URL in the `XSQLConfig.xml` file's connection definition. Of course, object/relational functionality only works when using Oracle with the Oracle JDBC driver.

XSQL Servlet: Access to JServ Process

I am running the demo `helloworld.xsql`. Initially I was getting the following error:

```
XSQL-007 cannot acquire a database connection to process page
```

Now my request times out and I see the following message in the `jserv/log/jserv.log` file:

```
Connections from Localhost/127.0.0.1 are not allowed
```

Is this a security issue? Do we have to give explicit permission to process an `.xsql` page. If so, how do we do that? I am using Apache web server and Apache jservlet and Oracle as database. I have Oracle client installed and `Tnsnames.ora` file configured to get database connection. My `XSQconnections.xml` file is configured correctly.

Answer

This looks like a generic JServ problem. You have to make sure that your `security.allowedAddresses=property` in `jserv.properties` allows your current host access to the JServ process where Java runs. Can you successfully run `*any*` JServ Servlet?

XSQL on Oracle8i Lite

I am trying to use XSQL with Oracle8i Lite (Win 98) and Apache/JServ Webserver. I am getting error message "no oljdbc40 in java.library.path" even though I have set the `olite40.jar` in my classpath (which contains the `POLJDBC` driver). Is there anything extra I need to do to run XSQL for Oracle8i Lite.

Answer

You must include the following instruction in your `jserv.properties` file:

```
wrapper.path=C:\orant\bin
```

where `C:\orant\bin` is the directory where (by default) the `OLJDBC40.DLL` lives.

Note that this is *not* `wrapper.classpath`, it's `wrapper.path`.

Handling Multi-Valued HTML Form Parameters

Is there any way to handle multi-valued HTML `<form>` parameters which is needed for `<input type="checkbox">`?

Answer

There is no built-in way, but you could use a custom Action Handler like this:

```
// MultiValuedParam: XSQL Action Handler that takes the value of
// ----- a multi-valued HTTP request parameter and
// sets the value of a user-defined page-parameter
// equal to the concatenation of the multiple values
// with optional control over the separator used
// between values and delimiter used around values.
// Subsequent actions in the page can then reference
// the value of the user-defined page-parameter.
import oracle.xml.xsql.*;
import javax.servlet.http.*;
import org.w3c.dom.*;
public class MultiValuedParam extends XSQLActionHandlerImpl {
    public void handleAction(Node root) {
        XSQLPageRequest req = getPageRequest();
        // Only bother to do this if we're in a Servlet environment
        if (req.getRequestType().equals("Servlet")) {
            Element actElt = getActionElement();
            // Get name of multi-valued parameter to read from attribute
            String paramName = getAttributeAllowingParam("name",actElt);
            // Get name of page-param to set with resulting value
            String pageParam = getAttributeAllowingParam("page-param",actElt);
            // Get separator string
            String separator = getAttributeAllowingParam("separator",actElt);
            // Get delimiter string
            String delimiter = getAttributeAllowingParam("delimiter",actElt);
            // If the separator is not specified or is blank, use comma
            if (separator == null || separator.equals("")) {
                separator = ",";
            }
        }
        // We're in a Servlet environment, so we can cast
        XSQLServletPageRequest spReq = (XSQLServletPageRequest)req;
        // Get hold of the HTTP Request
        HttpServletRequest httpReq = spReq.getHttpServletRequest();
        // Get the String array of parameter values
```

```

String[] values = httpReq.getParameterValues(paramName);
StringBuffer str = new StringBuffer();
// If some values have been returned
if (values != null) {
    int items = values.length;
    // Append each value to the string buffer
    for (int z = 0; z < items; z++) {
        // Add a separator before all but the first
        if (z != 0) str.append(separator);
        // Add a delimiter around the value if non-null
        if (delimiter != null) str.append(delimiter);
        str.append(values[z]);
        if (delimiter != null) str.append(delimiter);
    }
    // If page-param attribute not provided, default page param name
    if (pageParam == null) {
        pageParam = paramName+"-values";
    }
    // Set the page-param to the concatenated value
    req.setPageParam(pageParam,str.toString());
}
}
}
}
}

```

Then you can use this custom action in a page like this:

```

<page xmlns:xsql="urn:oracle-xsql">
  <xsql:action handler="MultiValuedParam" name="guy" page-param="p1" />
  <xsql:action handler="MultiValuedParam" name="guy" page-param="p2"
    delimiter="" />
  <xsql:action handler="MultiValuedParam" name="guy" page-param="p3"
    delimiter="&quot;" separator=" " />
  <xsql:include-param name="p1"/>
  <xsql:include-param name="p2"/>
  <xsql:include-param name="p3"/>
</page>

```

If this page is requested with the URL below, containing multiple parameters of the same name to produce a multi-valued attribute:

`http://yourserver.com/page.xsql?guy=Curly&guy=Larry&guy=Moe`

Then the page returned will be:

```
<page>
```

```

<p1>Curly,Larry,Moe</p1>
<p2>'Curly','Larry','Moe'</p2>
<p3>"Curly" "Larry" "Moe"</p3>
</page>

```

Of course you could also use the value of the multi-valued page parameter above in a SQL statement by doing:

```

<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:action handler="MultiValuedParam" name="guy" page-param="list"
    delimiter=" " />
  <!-- Above custom action sets the value of page param named 'list' -->
  <xsql:query>
    SELECT * FROM sometable WHERE name IN ( {@list} )
  </xsql:query>
</page>

```

XSQL Servlet and Oracle 7.3

Is there anything that prevents me from running the XSQL Servlet with Oracle 7.3? I know the XML SQL Utility can be used with Oracle 7.3 as long as I use it as a client-side utility.

Answer

No. Just make sure you're using the Oracle JDBC driver, which can connect to an Oracle 7.3 database with no problems.

Out Variable Not Supported in <xsql:dml>

I using <xsql:dml> to call a stored procedure which has one OUT parameter, but I was not able to see any results. The executed code results in the following statement:

```
<xsql-status action="xsql:dml" rows="0"/>
```

Answer

You cannot set parameter values by binding them in the position of OUT variables in this release using <xsql:dml>. Only IN parameters are supported for binding. You can create a wrapper procedure that constructs XML elements using the HTP package and then your XSQL page can invoke the wrapper procedure using <xsql:include-owa> instead.

For an example, suppose you had the following procedure:

```

CREATE OR REPLACE PROCEDURE addmult(arg1          NUMBER,
                                   arg2          NUMBER,
                                   sumval OUT NUMBER,
                                   prodval OUT NUMBER) IS

BEGIN
    sumval := arg1 + arg2;
    prodval := arg1 * arg2;
END;

```

You could write the following procedure to "wrap" it, taking all of the IN arguments that the procedure above expects, and then "encoding" the OUT values as a little XML datagram that you print to the OWA page buffer:

```

CREATE OR REPLACE PROCEDURE addmultwrapper(arg1 NUMBER, arg2 NUMBER) IS
    sumval NUMBER;
    prodval NUMBER;
    xml     VARCHAR2(2000);
BEGIN
    -- Call the procedure with OUT values
    addmult(arg1,arg2,sumval,prodval);
    -- Then produce XML that encodes the OUT values
    xml := '<addmult>' ||
           '<sum>' || sumval || '</sum>' ||
           '<product>' || prodval || '</product>' ||
           '</addmult>';
    -- Print the XML result to the OWA page buffer for return
    HTP.P(xml);
END;

```

This way, you can build an XSQL page like this that calls the wrapper procedure:

```

<page connection="demo" xmlns:xsql="urn:oracle-xsql">
  <xsql:include-owa bind-params="arg1 arg2">
    BEGIN addmultwrapper(?,?); END;
  </xsql:include-owa>
</page>

```

This allows a request like:

<http://yourserver.com/addmult.xsql?arg1=30&arg2=45>

to return an XML datagram that reflects the OUT values like this:

```

<page>
  <addmult><sum>75</sum><product>1350</product></addmult>
</page>

```

Unable to Connect Errors

Trying to play with XSQL I'm unable to connect to a database I get errors like this running the `helloworld.xsql` example:

```
Oracle XSQL Servlet Page Processor 9.0.0.0.0 (Beta)
XSQL-007: Cannot acquire a database connection to process page.
Connection refused(DESCRIPTION=(TMP=)(VSNNUM=135286784)(ERR=12505)
(ERROR_STACK=(ERROR=(CODE=12505)(EMFI=4))))
```

Does this mean that it has actually found the config file? I have a user with `scott/tiger` setup.

Answer

Yes. If you get this far, it's actually attempting the JDBC connection based on the `<connectiondef>` info for the connection named "demo", assuming you didn't modify the `helloworld.xsql` demo page.

By default the `XSQLConfig.xml` file comes with the entry for the "demo" connection that looks like this:

```
<connection name="demo">
  <username>scott</username>
  <password>tiger</password>
  <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
  <driver>oracle.jdbc.driver.OracleDriver</driver>
</connection>
```

So the error you're getting is likely because:

1. Your database is not on "localhost" machine.
2. Your database SID is not ORCL
3. Your TNS Listener Port is not 1521

Make sure those values are appropriate for your database and you should be in business.

Using Other File Extensions Besides *.xsql

I want users to think they are accessing HTML files or XML files with extensions `.html` and `.xml` respectively, however I'd like to use XSQL so serve the HTML and XML to them. Is it possible to have the XSQL Servlet recognize files with an extension of `.html` and/or `.xml` in addition to the default `.xsql` extension?

Answer

Sure. There is nothing sacred about the `*.xsql` extension, it is just the default extension used to recognize XSQL pages. You can modify your servlet engine's configuration settings to associate any extension you like with the `oracle.xml.xsql.XSQLServlet` servlet class using the same technique that was used to associate the `*.xsql` extension with it.

Avoiding Errors for Queries Containing XML Reserved Characters

I have a page like:

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT id, REPLACE(company, '&', 'and') company, balance
  FROM vendors
  WHERE outstanding_balance < 3000
</xsql:query>
```

but when I try to request the page I get an error:

```
XSQL-005: XSQL page is not well-formed.
XML parse error at line 4, char 16
Expected name instead of '
```

What's wrong?

Answer

The problem is that the ampersand character (&) and the less-than sign (<) are reserved characters in XML because:

- The & starts the sequence of characters that designates an entity reference like ` ` or `<`;
- The < starts the sequence of characters that designates an element like `<SomeElement>`

To include a literal ampersand character or less-than character you need to either encode each one as a entity reference like this:

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT id, REPLACE(company, '&amp;', 'and') company, balance
  FROM vendors
  WHERE outstanding_balance &lt;t; 3000
</xsql:query>
```


Alternatively, you can surround an entire block of text with a so-called CDATA section that begins with the sequence `<![CDATA[` and ends with a corresponding `]]>` sequence. All text contained in the CDATA section is treated as literal.

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
<![CDATA[
  SELECT id, REPLACE(company,'&','and') company, balance
  FROM vendors
  WHERE outstanding_balance < 3000
]]>
</xsql:query>
```

Using JDeveloper to Build Oracle XML Applications

This chapter contains the following sections:

- [Introducing JDeveloper9i](#)
- [What's Needed to Run JDeveloper9i](#)
- [XML in Business Components for Java \(BC4J\)](#)
- [Building XSQL Clients with Business Components for Java \(BC4J\)](#)
- [XML Features in JDeveloper9i](#)
- [Building XML Applications with JDeveloper](#)
- [Using JDeveloper's XML Data Generator Web Bean](#)
- [Using XSQL Servlet from JDeveloper](#)
- [Creating a Mobile Application in JDeveloper](#)
- [Frequently Asked Questions \(FAQs\): Using JDeveloper to Build XML Applications](#)

Introducing JDeveloper9i

Oracle JDeveloper9i is a J2EE™ development environment with end-to-end support for developing, debugging, and deploying e-business applications. JDeveloper empowers users with highly productive tools, such as the industry's fastest Java debugger, a new profiler, and the innovative CodeCoach tool for code performance analysis and improvement.

To take J2EE application development to a higher level of productivity, JDeveloper now offers Business Components for Java (BC4J), a standards-based, server-side framework for creating scalable, high-performance Internet applications. The framework provides design-time facilities and runtime services to drastically simplify the task of building and reusing business logic.

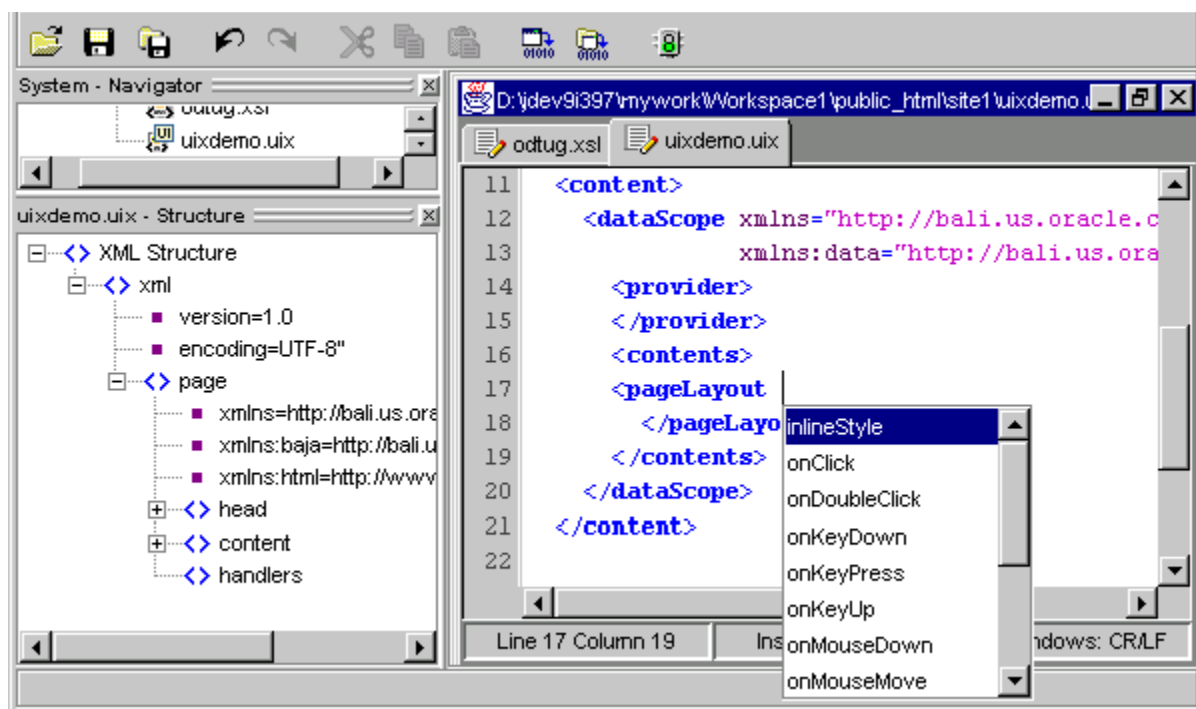
JDeveloper9i has a new schema-driven XML editor. See [Figure 11-1](#). An XML Schema Definition defines the structure of an XML document and is used in the editor to validate the XML and help developers when typing. This feature is called *Code Insight* and provides a list of valid alternatives for XML elements or attributes in the document. Just by specifying the schema for a certain language, the editor can assist you in creating a document in that markup language.

Oracle JDeveloper9i simplifies the task of working with Java application code and XML data and documents at the same time. It features drag-and-drop XML development modules. These include the following:

- Color-coded syntax highlighting for XML
- Built-in syntax checking for XML and Extensible Style Sheet Language (XSL)
- XSQL Pages and Servlet support, where developers can edit and debug Oracle XSQL Pages, Java programs that can query the database and return formatted XML, or insert XML into the database without writing code. The integrated servlet engine allows you to view XML output generated by Java code in the same environment as your program source, making it easy to do rapid, iterative development and testing.
- Includes Oracle's XML Parser for Java
- Includes XSLT Processor
- Related XDK for JavaBeans components
- XSQL Page Wizard. See "[Page Selector Wizard](#)" on page 11-11.
- XSQL Element Wizard. See "[XSQL Element Wizard](#)" on page 11-10.
- XSQL ActionHandlers

- Schema-driven XML editor.

Figure 11–1 JDeveloper9i's Schema-Driven XML Editor in Action



Oracle XML Developer's Kit (XDK) is integrated into JDeveloper, so that it offers many utilities to help Java developers handle, create, and transform XML. For example, when designing with XSQL Servlet, you can query and manipulate database information, generate XML documents, transform them using XSLT stylesheets, and make them available on the web.

See Also: [Chapter 10, "XSQL Pages Publishing Framework"](#)

Business Components for Java (BC4J)

Oracle Business Components for Java is a 100%-Java, XML-powered framework that enables productive development, portable deployment, and flexible customization of multi-tier, database-savvy applications from reusable business components.

Application developers use the Oracle Business Components framework and Oracle JDeveloper 's integrated design-time wizards, component editors, and productive Java coding environment to assemble and test application services from reusable business components.

These application services can then be deployed as either CORBA Server Objects or EJB Session Beans on enterprise-scale server platforms supporting Java technology.

The same server-side business component can be deployed without modification as either a JavaServer Pages/Servlet application or Enterprise JavaBeans component. This deployment flexibility, enables developers to reuse the same business logic and data models to deliver applications to a variety of clients, browsers, and wireless Internet devices without having to rewrite code.

In JDeveloper, you can customize the functionality of existing Business Components by using the new visual wizards to modify your XML metadata descriptions.

Oracle JDeveloper XML Strategy

Oracle JDeveloper9i supports building XML applications. JDeveloper new integrated schema-driven XML code editor works on XML Schema-based documents such as:

- For creating XML Schemas
- For creating XSLT Stylesheets,...

with “tag-insight” to help you easily enter the correct elements and attributes as defined by the schema. In addition to the editing capabilities, JDeveloper’s XML code editor also has the following features:

- Error highlighting
- Property inspection
- Tree-like view in the structure pane

Further Information About JDeveloper

See Also:

- <http://otn.oracle.com/products/jdev/>
- *Oracle9i Java Developer's Guide*
- *Oracle JavaServer Pages Developer's Guide and Reference*
- *Oracle9i CORBA Developer's Guide and Reference*
- *Oracle9i Enterprise JavaBeans Developer's Guide and Reference*
- *Oracle9i Java Stored Procedures Developer's Guide*
- *Oracle9i Java Tools Reference*
- *Oracle9i JDBC Developer's Guide and Reference*
- *Oracle9i JPublisher User's Guide*
- *Oracle9i Oracle Servlet Engine User's Guide*
- *Oracle9i SQLJ Developer's Guide and Reference*

What's Needed to Run JDeveloper9i

JDeveloper9i is an IDE that has been written in Java and therefore, runs on Windows NT, Windows 2000, Linux and Solaris operating systems. It needs a minimum of 128 Mb RAM.

Minimum system requirements for JDeveloper

Refer to JDeveloper Release Notes. As more products are run on the same machine, system requirements are increased. A typical development environment for running JDeveloper includes:

- Running JDeveloper
- Running Oracle locally
- Running Oracle Application Server locally
- Additional third party tools (profilers, version control, modelers,...)

These add to system requirements, in terms of actual CPU usage and in disk space needs.

Accessing JDeveloper9i

The beta release of JDeveloper9i will be available in the summer of 2001 from the Oracle Technology Network (OTN) at <http://otn.oracle.com>.

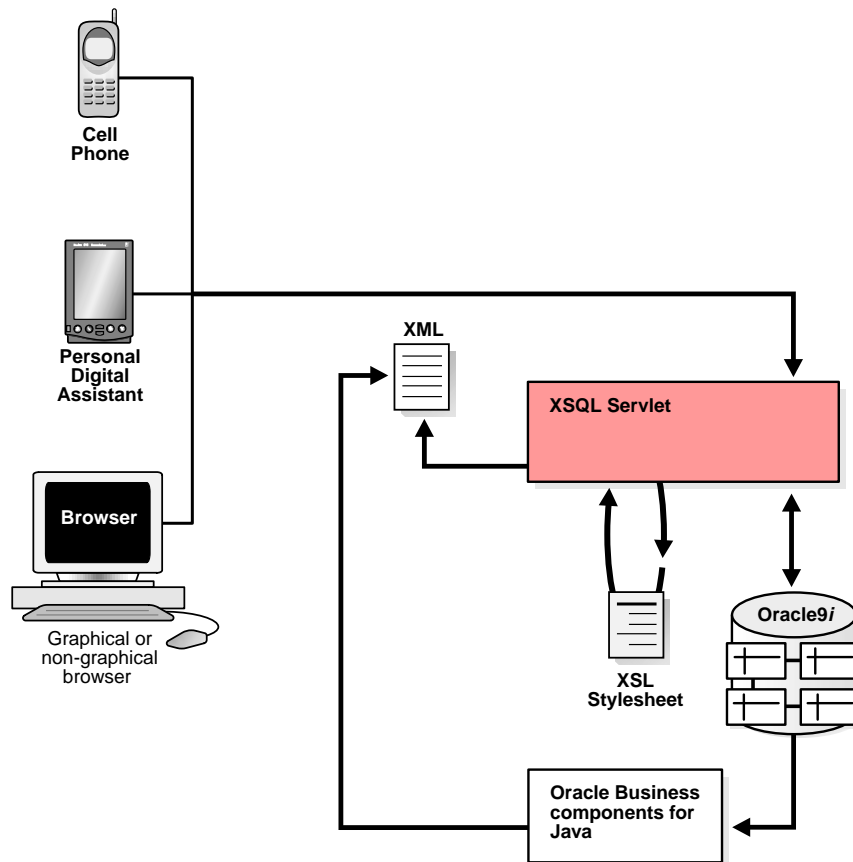
XML in Business Components for Java (BC4J)

The Business Components for Java (BC4J) framework in JDeveloper9i uses XML to define the metadata that represents the declarative settings and features of the objects. Custom or complex business logic can be implemented in Java.

- The BC4J Tester enables you to see data in view objects as XML.
- Business rules, such as validation rules, are stored in XML rather than Java source code
- Easy customization of business applications by changing XML rather than Java source code
- Applications are easier to read and understand by abstracting the logic in XML

BC4J uses XML to Store Metadata The business components for Java framework that ships with JDeveloper uses XML to store metadata about its application components. Important information is now stored in a structured document rather than in Java source code. This makes the application easier to understand and customize. The application is now customizable without having access to the source code. [Figure 11-2](#) shows how BC4J is used with XSQL servlet to generate XML documents.

See Also: <http://otn.oracle.com/products/bc4j/>

Figure 11-2 Using Business Components for Java (BC4J)

Business rules can be changed on site without needing access to the underlying component source code.

Building XSQL Clients with Business Components for Java (BC4J)

In JDeveloper 9i, you can build XSQL Pages which can integrate with BC4J application modules and thereby serve application logic from the middle tier to multiple clients. You can retrieve XML data and present it to any kind of a client device just by applying the corresponding stylesheet.

The following features will assist you in building XSQL clients with BC4J:

- Object Gallery
- XSQL Element Wizard
- Page Selector Wizard

Note: These appearance of these features may differ in the JDeveloper9i production version.

See Also: [Chapter 12, "Building BC4J and XML Applications"](#)

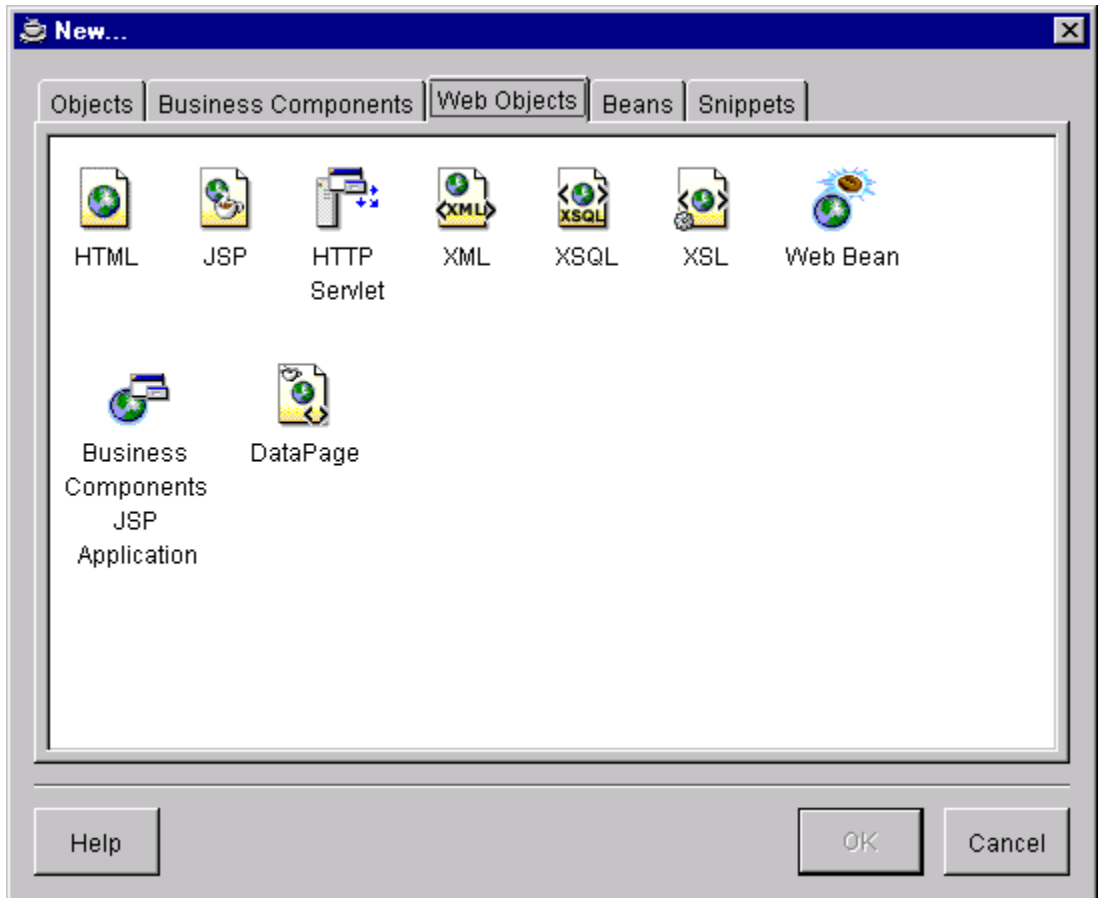
Object Gallery

The Web Object Gallery has icons to assist in creating XSQL, XML, and XSL documents easily. When you click on them, the basic tags for these pages are generated and you can then enhance them.

The XSQL Pages icon is of special interest because the XSQL Element Wizard can be used, after generating your basic XSQL pages, to insert data bound tags in the XSQL pages. [Figure 11-3](#) illustrates JDeveloper's Object Gallery.

See Also: ["XSQL Element Wizard"](#) on page 11-10.

Figure 11-3 JDeveloper's Object Gallery Showing the new XSQL, XML, and XSL Icons

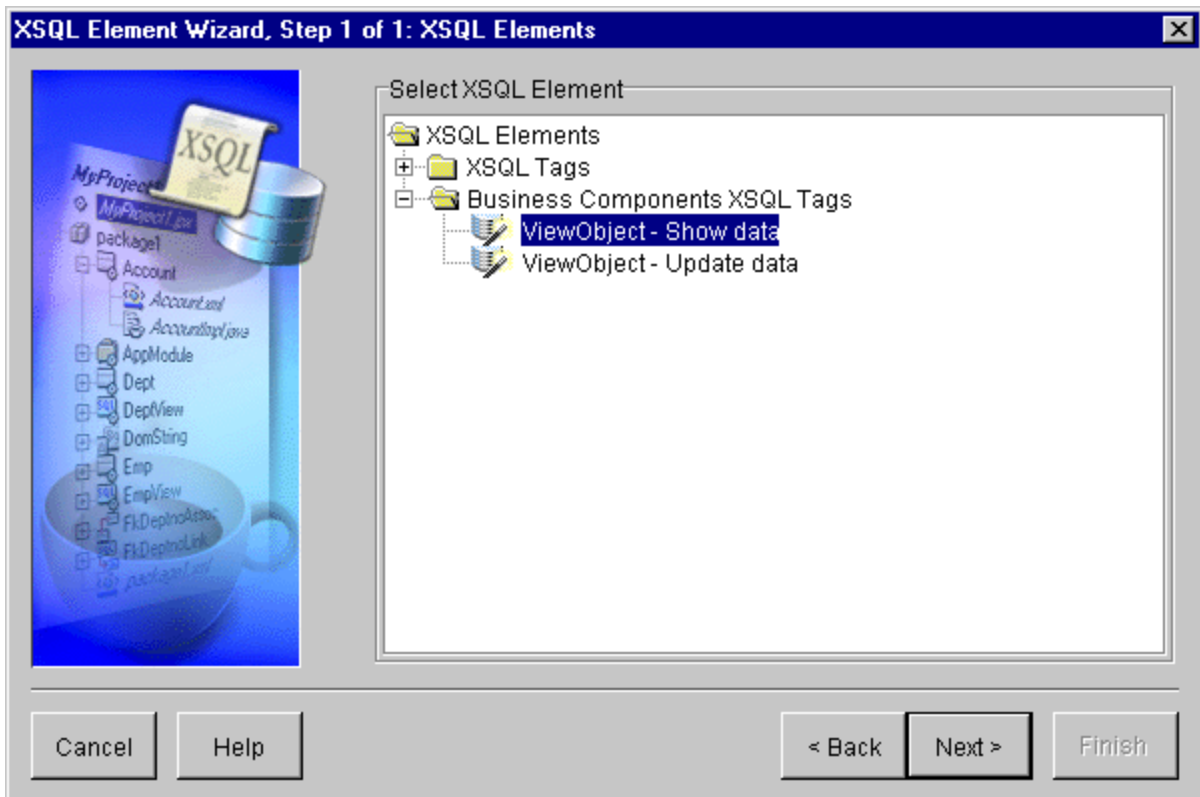


Note: The appearance of these features (wizards) may change in the production release.

XSQL Element Wizard

XSQL Element Wizard provides you with a mechanism to add tags which allows accessing database tables or BC4J View Objects. You can either perform queries against them or update the underlying database tables through them. [Figure 11-4](#) illustrates the JDeveloper9i XSQL Element Wizard.

Figure 11-4 JDeveloper's XSQL Element Wizard



Note: The appearance of these features (wizards) may change in the production release.

Page Selector Wizard

When you need create XSQL pages while building a web application, you can invoke Page Wizard which allows you to create XSQL Pages on top of either database tables directly or on top of BC4J View Objects. When you choose to build an XSQL Page on top of a BC4J View Object, you are prompted to select an application module from a list or create a new application module and then build the XSQL Pages based application.

See Also: *Oracle9i Java Developer's Guide*

XML Features in JDeveloper9i

The following lists JDeveloper9i's supported Oracle XML Developer's Kit for Java (XDK for Java) components:

- Oracle XML Parser for Java
- Oracle XSQL Servlet

You can use the XML Parser for Java including the XSLT Processor and the XML SQL Utility in JDeveloper as all these tools are written in Java. JDeveloper provides these components.

Sample programs which demonstrate how to use these tools can be found in the `[JDeveloper]/Samples/xmlsamples` directory.

Oracle XDK and Transviewer Beans Integration

Oracle XDK for Java consists of the following XML tools:

- XML Parser for Java
- XML- SQL Utility for Java
- XML Java Class Generator
- XSQL Servlet
- XML Transviewer Beans

All these utilities are written in Java and hence can easily be dropped into JDeveloper and used 'out of box'. You can also update the XDK for Java components with the latest versions downloaded from Oracle Technology Network (OTN) at <http://technet.oracle.com/tech/xml>.

Oracle XDK for Java also includes the XML Transviewer Beans. These are a set of Java Beans that permit the easy addition of graphical or visual interfaces to XML applications. Bean encapsulation includes documentation and descriptors that make them accessible directly from JDeveloper. You can drop these beans into the TOOLS palette and use them to build applications such as XML/XSL editors.

See Also: [Chapter 23, "Using XML Transviewer Beans"](#) for more information on how to use the Transviewer Beans.

Oracle XML Parser for Java

Including the Oracle XML Parser for Java in your project allows you to write applications that can search and process XML documents. You can include the Oracle XML Parser in your project with one click as JDeveloper has a built-in library for it.

Code Insight makes understanding and using the code easier and in-place access to JavaDoc on the classes for reference. The XML parser for Java facilitates processing an XML document using either of the following interfaces:

- DOM: a tree of W3C DOM
- SAX: a stream of SAX events

Oracle XSQL Servlet

The XSQL Servlet is a tool that processes SQL queries and outputs the result set as XML. This processor is implemented as a Java servlet and takes as its input an XML file containing embedded SQL queries. It uses the XML Parser for Java and the XML SQL Utility to perform many of its operations.

The XSQL Servlet offers a productive and easy way to get XML in and out of the database. Using simple scripts you can:

- Generate simple and complex XML documents
- Apply XSL Stylesheets to generate into any text format
- Parse XML documents and store the data in the database
- Create complete dynamic web applications without programming a single line of code

JDeveloper XSQL Example 1: emp.xsql

For example, consider the following XML example:

```

<?xml version="1.0"?>
<xml-stylesheet type="text/xsl" href="emp.xsl"?>
<FAQ xmlns:xsql="urn:oracle-xsql" connection = "scott">
  <xsql:query doc-element="EMPLOYEES" row-element="EMP">
    select e.ename, e.sal, d.dname as department
    from dept d, emp e
    where d.deptno = e.deptno
  </xsql:query>
</FAQ>

```

Generates the following:

```

<EMPLOYEES>
  <EMP>
    <ENAME>Scott</ENAME>
    <SAL>1000</SAL>
    <DEPARTMENT>Boston</DEPARTMENT>
  </EMP>
  <EMP>
    ...
</EMPLOYEES>

```

With JDeveloper9i you can easily develop and execute XSQL files. The built in Web Server and the user's default Web Browser will be used to display the resulting pages.

Using ActionHandlers in XSQL Pages

XSQL ActionHandlers are Java classes which can be invoked from XSQL Page applications very easily. Since these are Java classes they can be debugged from JDeveloper just like any other Java application.

If you are building an XSQL Pages application, you can make use of the XSQL Action Handler to extend the set of actions that can be performed to handle more complex jobs. You will need to debug this ActionHandler.

Your XSQL Pages should be in the directory specified in the Project Property "HTML Paths" settings for "HTML Source Directory".

To debug your ActionHandler carry out these steps:

1. Assume you have created an .xsql file which has reference to a custom ActionHandler called MyActionHandler.
2. Debug this ActionHandler because it is not exactly behaving as you expect.
3. Set breakpoints in your Java source file.

4. He right mouse clicks on the .xsql file and now chooses Debug... from the menu.

See Also: *The JDeveloper Guide* under the online HELP menu.

XML Data Generator Web Bean

Oracle JDeveloper has an XML Data Generator Web Bean. It generates XML containing the data from a View Object and renders it to the output stream of a JSP response.

You can author JSP pages that use XML and XSL to render a response to the client.

This XML Web Bean can be used in JSP and Servlet applications. It reads data from a Business Component (View Object) and produces the appropriate XML. The strength of this Web Bean is that it analyzes the Business Component Application and navigates through it's hierarchy to produce the nested XML.

The XML Web Beans also allows the specification of an XSL Stylesheet. In addition to XML, the Web Bean can then generate HTML, WML, transformed XML and any other text format.

Mobile Application Development with Portal-To-Go and JDeveloper

Portal-To-Go and Oracle JDeveloper together offer an extremely powerful environment for developing mobile applications. Developers can use JDeveloper to generate XML from the database or from a Business Components for Java Application and use Portal-To-Go to deliver content to Web browsers, PDAs, or Cell phones.

Building XML Applications with JDeveloper

Consider the following example that demonstrates how XML is used to represent data, not present it. It shows the many to one relationship between employees and departments.

JDeveloper XML Example 1: BC4J Metadata

```
<Departments>
<Dept>
  <Deptno>10</Deptno>
  <Dname>Sales</Dname>
  <Loc>
```



```

<Employees>
  <Employee>
    <Empno>1001</Empno>
    <Ename>Scott</Ename>
    <Salary>80000</Salary>
  </Employee>
</Employees>
...
  </Employee>
</Employees>
</Dept>
<Dept>
...

```

Procedure for Building Applications in JDeveloper9i

To build this project in JDeveloper9i carry out the following steps:

1. Start a New JDeveloper Project by selecting File > New Project.
2. Create a Business Components for Java application.
3. Create an XSQL Page based upon a BC4J application module, by invoking the Page Selector Wizard.
4. Select the application module from the list that pops up.
5. Select the View Object on which you want to base your XSQL Page.
6. Select the columns that you want to view.

When you finish these steps in the Page Wizard, you should have an XSQL Page based on the Business Components for Java (BC4J) framework View objects. When you run this page, it sends the XML data to your browser. You could optionally create a stylesheet to format the data so that it appears in a way that you prefer or you can tune it so that it can be displayed on a PDA or cellphone.

Using JDeveloper's XML Data Generator Web Bean

The XML Data Generator Web (Bean) can be used in JSP and Servlet applications. It reads data from a Business Component (View Object) and produces the appropriate XML. The strength of this Web Bean is the following:

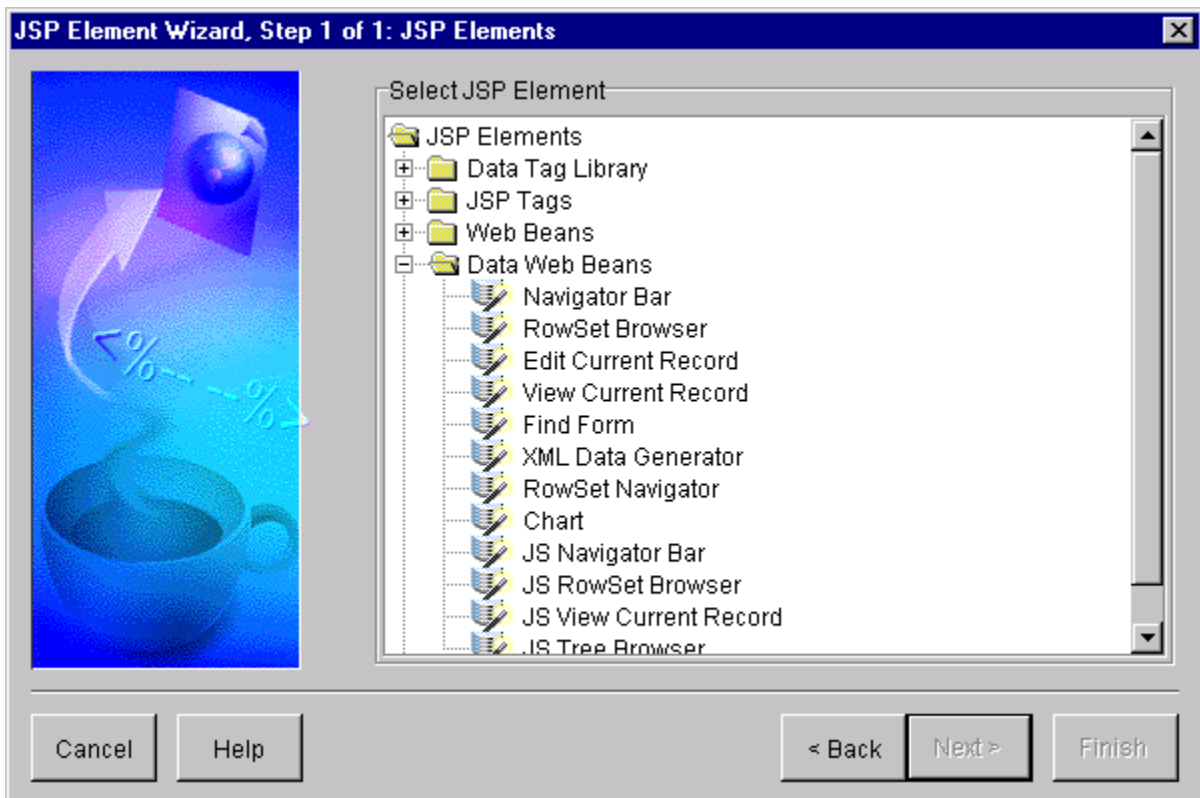
- It analyzes the Business Component Application and navigates through it's hierarchy to produce the nested XML.

- It allows specification of an XSL Stylesheet. The Web Bean can then generate HTML, WML, transformed XML, and any other text format.

The Data Generator Web Bean is in the “Data Web Beans” category of the JSP Elements wizard. [Figure 11-5](#) illustrates accessing the XML Data Generator Web Bean from the JSP Element Wizard.

Call the Element Wizard from your JSP or XSQL Page by right-clicking anywhere on the Page where you want to include an element. Specify the stylesheet as a parameter in the Element Wizard.

Figure 11-5 JSP Element Wizard: XML Data Generator Bean

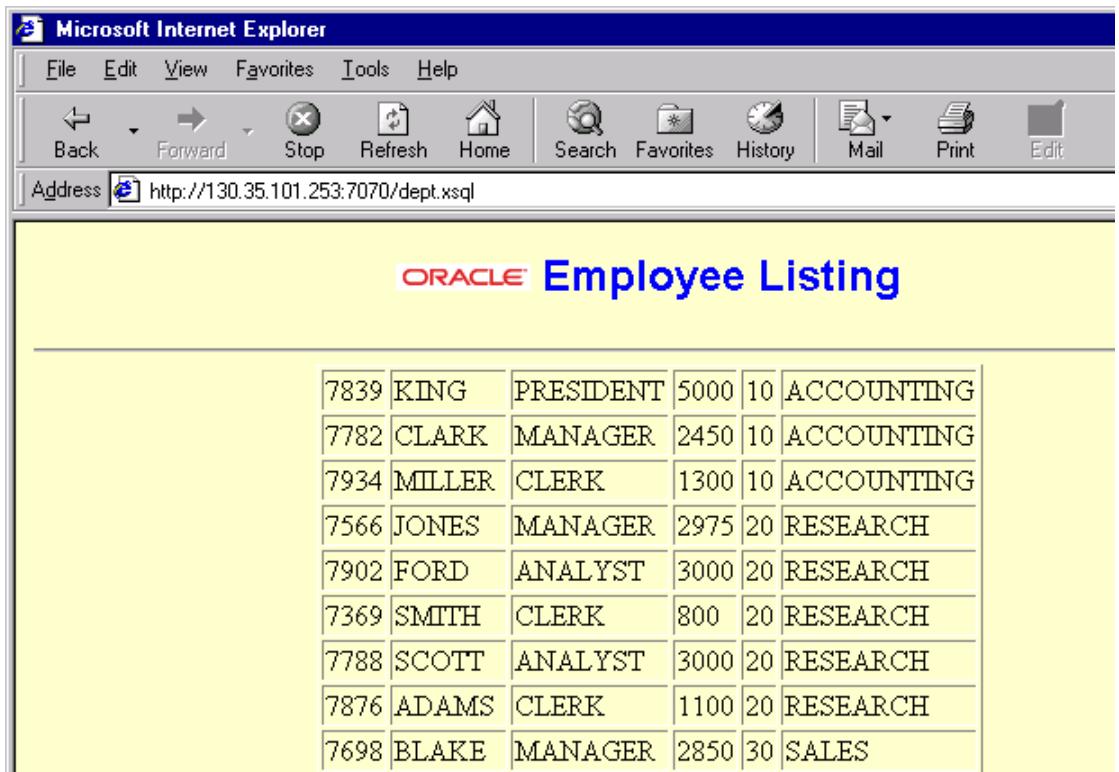


Note: The appearance of these features (wizards) may change in the production release.

Once you launch this wizard, you can specify a stylesheet to apply to the XML data that you generate and see the result of your output.

Figure 11–6 is an example output displayed by applying an XSL stylesheet to the employee listing.

Figure 11–6 Browser HTML Display Showing the Employee Listing (XML+XSLT=HTML)



Using XSQL Servlet from JDeveloper

XSQL Servlet offers a productive and easy way to get XML in and out of the database.

See Also: [Chapter 3, "Oracle XML Developer Kits \(XDKs\) and Components: Overview and General FAQs"](#) and [Chapter 10, "XSQL Pages Publishing Framework"](#) for information about how to use XSQL Servlet.

When using XSQL Servlet in JDeveloper, you do not need to include the XSQL Runtime in your project as this is already done for any new XSQL Page or XSQL wizard-based application.

Using simple scripts you can do the following from JDeveloper:

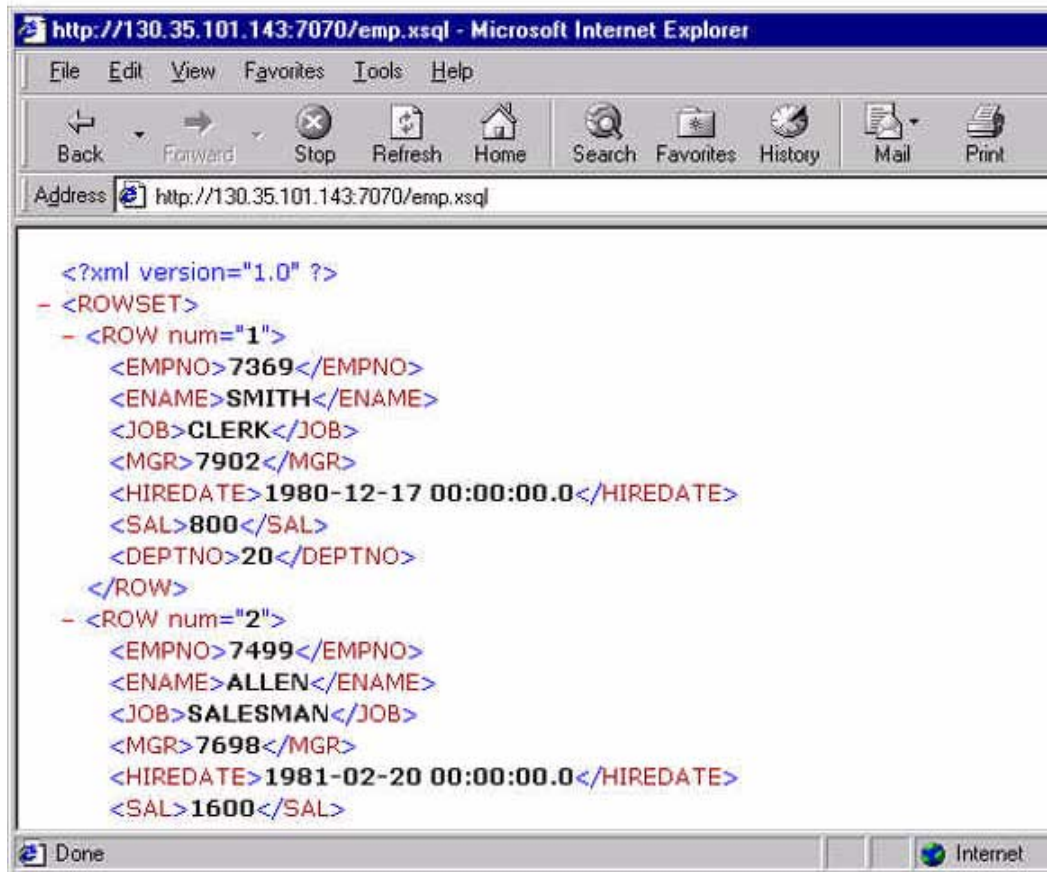
- Generate simple and complex XML documents
- Apply XSL stylesheets to generate into any text format
- Parse XML documents and store the data in the database
- Create complete dynamic web applications without programming a single line of code

Consider a simple query in an XSQL file, which returns details about all the employees in the emp table. The XSQL code to get this information would be as shown in Example 2.

JDeveloper XSQL Example 2: Employee Data from Table emp: emp.xsql

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
  select *
  from emp
  order by empno
</xsql:query>
```

[Figure 11-7](#) shows what the raw employee XML data displayed on the browser.

Figure 11–7 Employee Data in Raw XML Format

If you want to output your data in a tabular form as shown in [Figure 11–6](#), make a small modification to your XSQL code to specify a stylesheet. The changes you would make in this example are shown below highlighted.

JDeveloper XSQL Example 3: Employee Data with Stylesheet Added

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
  select *
  from emp
```

```
        order by empno  
</xsql:query>
```

The result would be like the table shown in [Figure 11-6](#). You can do a lot more with XSQL Servlet of course.

See Also: [Chapter 10, "XSQL Pages Publishing Framework"](#) and also the XDK for Java, XSQL Servlet Release Notes on OTN at <http://technet.oracle.com/tech/xml>

Creating a Mobile Application in JDeveloper

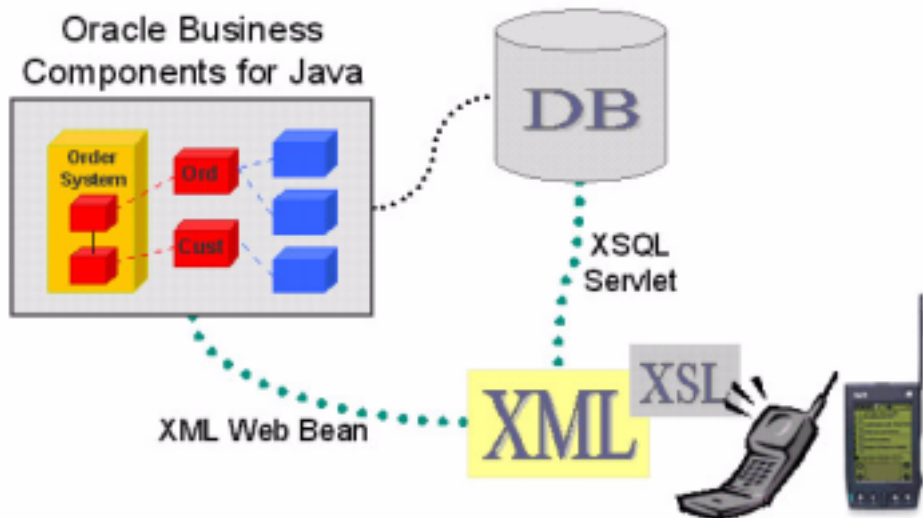
This mobile application is a Departments database application that demonstrates how Business Components for Java (BC4J) and XML can be used to develop applications that can be accessed over wireless devices. The application consists of two main parts:

- Server-side business logic which is developed using the Business Components for Java (BC4J) Framework and the second is the client part. The business logic consists of a view object based on the DEPT table in SCOTT's schema.
- A mechanism to query the DEPT table and update it from any client device including a browser, a cellular phone and a Palm Pilot. For the latter device, the application uses emulators running on Windows NT.

[Figure 11-8](#) shows schematically how the mobile application works with BC4J, XSQL Servlet, XSL Stylesheets, and Oracle9i.

You can see a more comprehensive demo of a similar application on <http://otn.oracle.com/tech/xml>

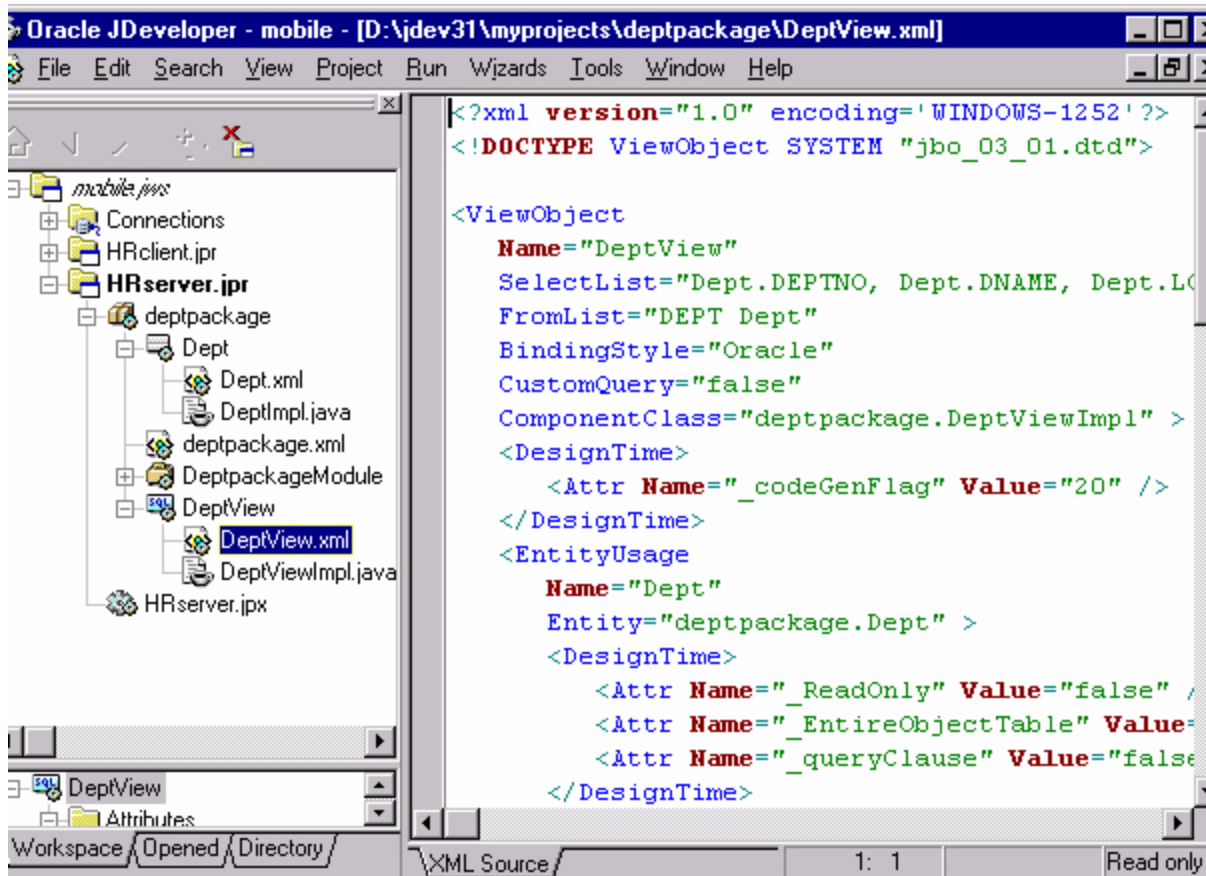
Figure 11–8 *Creating a Mobile Application in JDeveloper Using BC4J and XSQL Servlet*



1 Create the BC4J Application

First create the BC4J application. It connects to the SCOTT schema on an Oracle9i database. [Figure 11–9](#) shows the XML file containing the metadata about the DEPT object. See "[JDeveloper XML Example 1: BC4J Metadata](#)" on page 11-14.

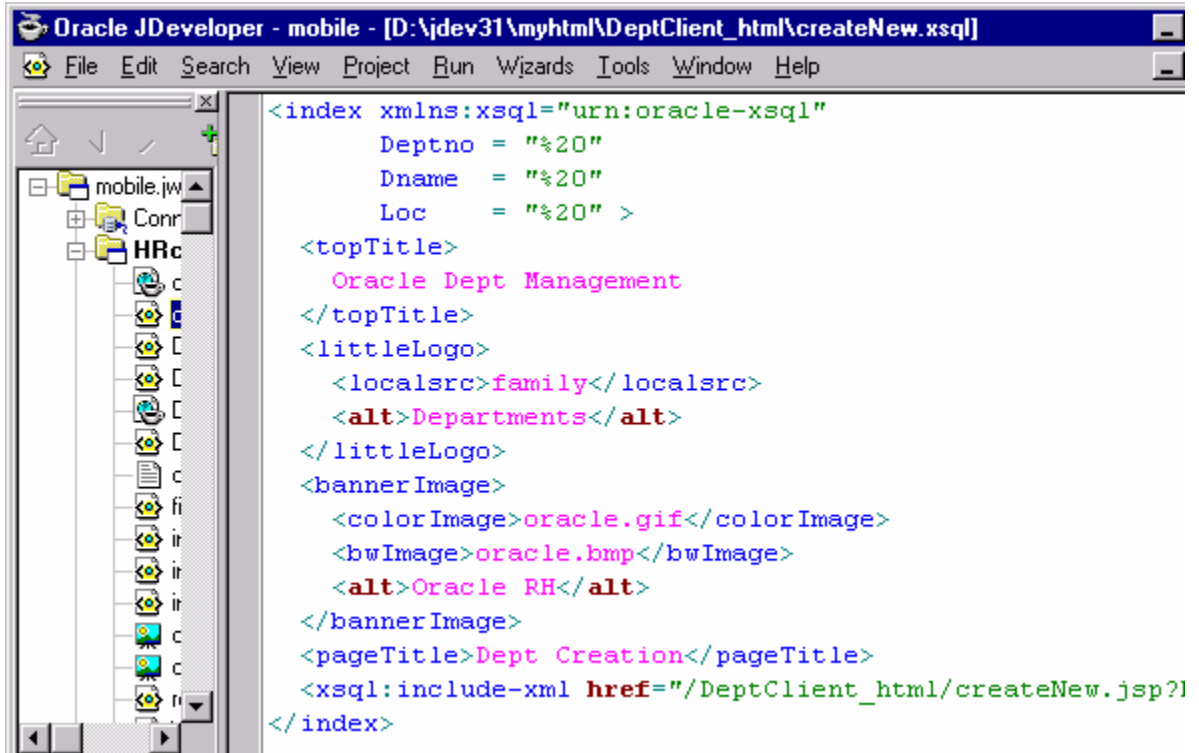
Figure 11–9 BC4J Application: DEPT View Object XML File



2 Create JSP Pages Based on a BC4J Application

You can then create JSP pages based upon this BC4J application. In the JSP pages you are introduced to the XML Data Generator Web Beans. Figure 11–10 shows the XSQL file which calls the JSP page to create the new department.

Figure 11–10 BC4J Application: XSQL File Calling JSP Page



3 Create XSLT Stylesheets According to the Devices Needed to Read The Data

We create XSLT stylesheets to go with the various client devices that we are going to access our data from. In your XSQL files, you specify the list of stylesheets and the protocols they go with which basically ties the stylesheets to the client device.

Figure 11–11 shows an example code snippet of a stylesheet (indexPP.xsl) which transforms the XML data to HTML for displaying on a browser on the Palm Pilot emulator.

Figure 11–11 BC4J Application: XSL Stylesheet (indexPP.xsl)

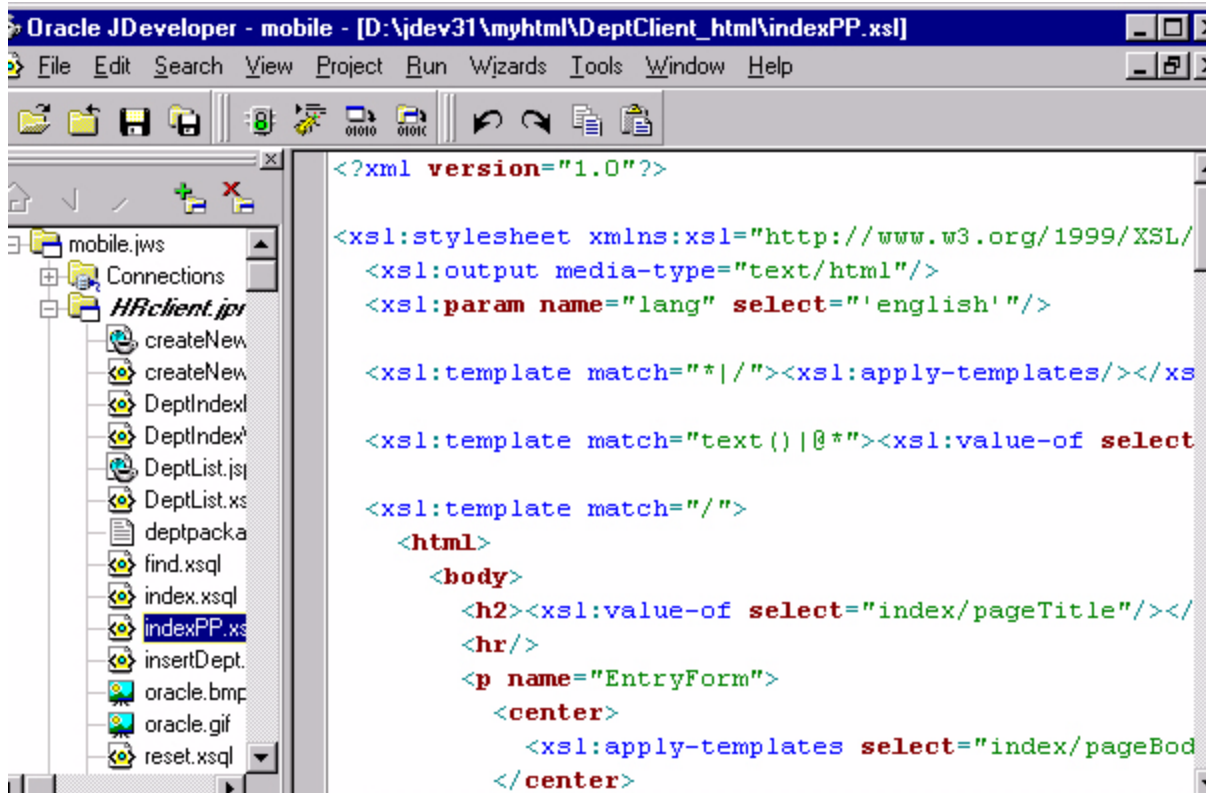


Figure 11–12 shows the Cell Phone Emulator running the Departments Application Client. It also shows the setup screen for the Cell Phone Emulator.

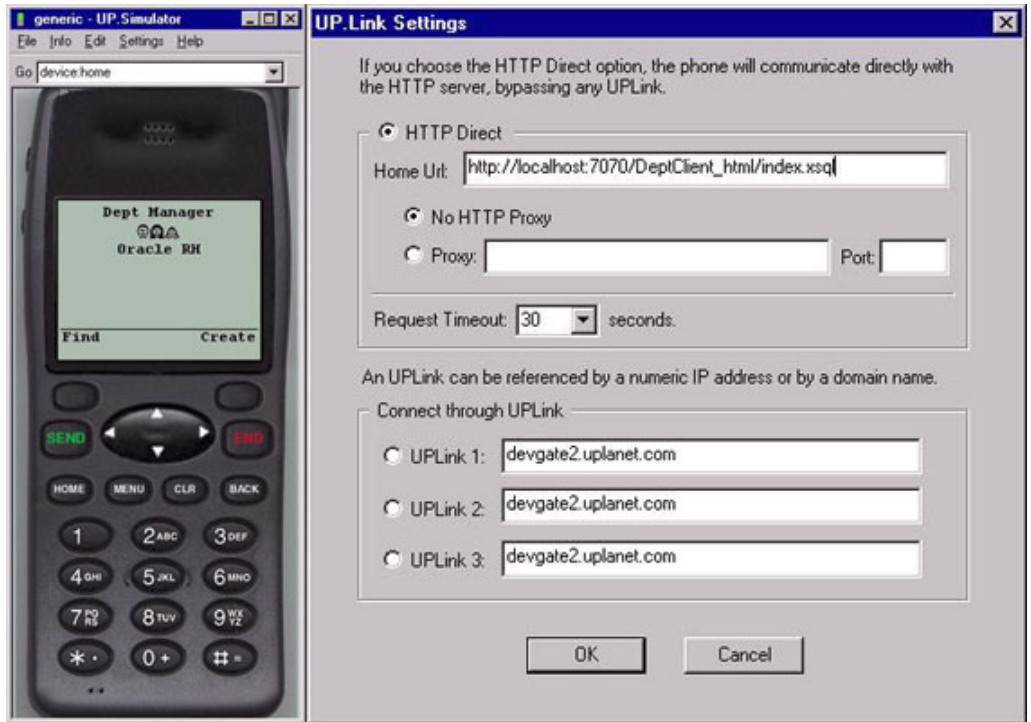
Figure 11–12 Cell Phone Emulator Running the Department Application Client

Figure 11–13 shows the Palm Pilot Emulator accessing the Departments Application via HandWeb Browser.

Figure 11–13 Palm Pilot Emulator Accessing the BC4J Departments Application Through HandWeb Browser



Frequently Asked Questions (FAQs): Using JDeveloper to Build XML Applications

Constructing an XML Document in JSP

I am dynamically constructing an XML document in a JSP page (from the results of data returned by a PL/SQL API) using the classes generated by the Class generator (based on a DTD) and then applying a XSL stylesheet to render the transformation in HTML. I see that this works fine only for the first time, i.e when the JSP is first accessed (and internally compiled), and fails every time the same page is accessed thereafter.

The error I get is:

```
"oracle.xml.parser.v2.XMLDOMException: Node doesn't belong to the current document"
```

The only way to make it work again is to compile the JSP, by just 'touching' the JSP page. Of course, this again works only once. I am using Apache JServ.

How can this be overcome? Does the 'static' code in the Java class generated for the top level node have to do anything with it?

Answer

It seems to me that you may have stored some "invalid" state in your JSP. And the XML Parser picks this "invalid" state, then, throws the exception you mentioned.

As far as I know, CRM does not use an HTTP session in their application. I guess this is true in your case also. You may have used a member variable to store some "invalid" state unintentionally. Member variables are the variables declared by the following syntax:

```
<%! %>
```

For example:

```
<%! Document doc=null; %>
```

Many JSP users misunderstand that they need to use this syntax to declare variables. In fact, you do not need to do that. In most of cases, you do not need a member variable. Member variables are shared by all requests and are initialized only once in the lifetime of the JSP.

Most users need stack variables or method variables. These are created and initialized per request. They are declared as a form of scriptlet as shown in the following example:

```
<% Document doc=null; %>
```

In this case, every request has its own "doc" object, and the doc object is initialized to null per request.

If you do not store an "invalid" state in session or method variables in your JSP, then there may be other reasons that cause this.

Using XMLData From BC4J

I am using XmlData to retrieve data from a BC4J. I Do not use XmlData from a JSP, but from a standalone java application. In the record I target, I have the value 'R & D'.

XmlData returns 'R & D', which is fine for HTTP, but not for my needs. Can XmlData not escape the characters, and just return what's in the database?

Answer

XmlData builds an in-memory DOM, so it must be the XML parser's serialization that's doing this. The only way I know is to do the following:

1. Write your own serializer for the DOM tree that does what you want.
2. Do an identity transform augmented with one template to write that data with `disable-output-escaping="yes"`

Running XML Parser for Java in JDeveloper 3.0

I have downloaded JDeveloper on my laptop (Windows 95 operating system). I am trying to run a sample XML parser program (SimpleParse.java). This program imports `org.w3c.dom.Document` class. I have set CLASSPATH in `autoexe.bat` with correct directory. The program runs on DOS prompt with `"java SimpleParse <filename>"` command. I am trying to run the same program through JDeveloper but it gives me following error:

```
"identifier org.w3c.dom.Document not found"
```

Am I missing something?

Answer

Make sure to include the Library named:

"Oracle XML Parser 2.0"

is in your project. This library is pre-defined with JDev 3.0 and higher and you just need to visit the Project | Properties... and look at the "Paths" tab to see your project's library list.

Click the (Add...) button and pick the above library from the list.

The `org.w3c.dom.*` interfaces are included in this Jar. They come from the W3C and define the Document Object Model standard API's for working with a tree of XML nodes.

Question 2

Now, if I wish to use the `@code` as a key, I use

```
<xsl:template match="aTextNode">
```

```
...
<xsl:param name="labelCode" select="@code"/>
  <xsl:value-of
select="document('messages.xml')/messages/msg[@id=$labelCode and
@lang=$lang]"/>
...
</xsl:template>
```

that works too, but I was wondering if there isn't a way to use the '@code' directly in the 'document()' line?

Answer 2

This is what the current() function is useful for. Rather than:

```
<xsl:param name="labelCode" select="@code"/>
<xsl:value-of
select="document('messages.xml')/messages/msg[@id=$labelCode and
@lang=$lang]"/>
```

you can do:

```
<xsl:value-of
select="document('messages.xml')/messages/msg[@id=current()/@code
and @lang = $lang]"/>
```

Question 3

And finally, another question: it is - or will it be - possible to retrieve the data stored in messages.xml from the database? How is the 'document()' instruction going to work where listener and servlet will run inside the database?

Answer 3

Sure. By the spec, the XSLT engine should read and cache the document referred to in the document() function. It caches the parsed document based on the string-form of the URI you pass in, so here's how you can achieve a database-based message lookup:

1. CREATE TABLE MESSAGES (lang VARCHAR2(2), code NUMBER, message VARCHAR2(200));
2. Create an xsql page like "msg.xsql" below:

```
<xsql:query lang="en" xmlns:xsql="urn:oracle-xsql" connection="demo"
```

```

        row-element="" rowset-element="">
select message
  from messages
  where lang = '{@lang}'
     and code = {@code}
</xsql:query>

```

3. Create a stylesheet that uses msg.xsql in the document() function like:

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
  <html><body>
    In English my name is
    <xsl:call-template name="msg">
      <xsl:with-param name="code">101</xsl:with-param>
    </xsl:call-template><br/>
    En espanol mi nombre es
    <xsl:call-template name="msg">
      <xsl:with-param name="code">101</xsl:with-param>
      <xsl:with-param name="lang">es</xsl:with-param>
    </xsl:call-template><br/>
    En fran&#231;ais, je m'appelle
    <xsl:call-template name="msg">
      <xsl:with-param name="code">101</xsl:with-param>
      <xsl:with-param name="lang">fr</xsl:with-param>
    </xsl:call-template><br/>
    In italiano, mi chiamo
    <xsl:call-template name="msg">
      <xsl:with-param name="code">101</xsl:with-param>
      <xsl:with-param name="lang">it</xsl:with-param>
    </xsl:call-template>
  </body></html>
</xsl:template>
<xsl:template name="msg">
  <xsl:param name="lang">en</xsl:param>
  <xsl:param name="code" />
  <xsl:variable name="msgurl"
select="concat('http://xml/msg.xsql?lang=', $lang, '&code=' , $code)"/>
  <xsl:value-of select="document($msgurl)/MESSAGE"/>
</xsl:template>
</xsl:stylesheet>

```

4. Try it out at <http://xml/testmessage.xsql>

This is great if you want to fetch the message from over the web. Alternatively, you could use the msg.xsql above but include it in your XSQL Page if that makes sense using:

```
<xsql:include-xsql href="msg.xsql?lang={@lang}&code={@code}" />
```

Or you could write your own custom action handler to use JDBC to fetch the message and include it in the XSQL page yourself.

Moving Complex XML Documents to a Database

I am moving XML documents to an Oracle database. The documents are fairly complex. Can an XML document and the Oracle Developer's Kit (XDK) generate a possible DDL format for how the XML Document should be stored in the database, ideally generating an Object-Relational Structure. Does anyone know of a tool that can do this?

Answer a

The best way may be to use the Class Generator. Use XML SQL Utility if DTD files are not already created. You'll still have to write a mapping program.

Another method is to create views and write stored procedures to update multiple tables. Unfortunately, you'll have to create your tables and views beforehand in either case.

BC4J Business Components for Java (BC4J) framework provides a general, meta-data-driven solution for mapping E-Commerce XML Messages into and out of the database. BC4J has a technical white paper on its features available at <http://otn.oracle.com/products/jdev/info/techwp20/wp.html>.

It is a Pure-Java, XML-Based business components framework for making building E-Commerce applications easier. It is a Java framework usable on its own, but also has tight development support built-into JDeveloper 3.0 IDE, available for download from <http://otn.oracle.com/software/>.

BC4J lets you flexibly map hierarchies of SQL-based "view components" to underlying business components that manage all application behavior (rules and processes) in a uniform way. It also supports *dynamic* functionality, so most of its features can be driven completely off XML metadata. You can build a layer which flexibly maps any XML Document into and out of the database using this framework. One key benefit is that when XML Documents are put into the system, they automatically can have all the same business rules validated.

Building BC4J and XML Applications

This chapter contains the following sections:

- [Introducing Business Components for Java \(BC4J\)](#)
- [BC4J Features](#)
- [Building BC4J XML Applications in JDeveloper](#)

Introducing Business Components for Java (BC4J)

Oracle Business Components for Java (BC4J) is a Java, XML-powered framework that enables development, portable deployment, and flexible customization of multi-tier, database applications from reusable business components.

Maps XML Messages

The Business Components for Java (BC4J) framework provides a general, metadata-driven solution for mapping E-commerce XML Messages into and out of the database.

Test BC4J Applications using JDeveloper

You can use Oracle BC4J framework and Oracle JDeveloper 's wizards and component editors to assemble and test application services from your reusable business components.

In JDeveloper, you can also customize the functionality of existing Business Components by using the visual wizards to modify your XML metadata descriptions.

See Also:

- [Chapter 11, "Using JDeveloper to Build Oracle XML Applications"](#)
- *Oracle9i Java Developer's Guide*
- <http://otn.oracle.com/products/bc4j>

BC4J Features

BC4J features include the following:

- Write Once, Deploy Anywhere
- A 100% Java application framework for building component-based, multitier enterprise applications
- Easily separate business logic and user interface (UI)
- Easily customize using extensible markup language (XML) and Java
- Helps automate complex database interactions

- Allows you to flexibly map hierarchies of SQL-based “view components” to underlying business components that manage all application behavior (rules and processes) uniformly.
- Supports *dynamic* functionality, such that features can be driven from XML metadata.
- You can build a layer which maps any XML document in and out of the database using this framework; when these XML Documents are used in the system, they will automatically have all the same business rules validated.

BC4J Advantages

BC4J has the following advantages:

- Reusable business logic
- Entities encapsulate business logic
- Associate and compose business components
- Flexible, updateable SQL-based views
- Optimized data retrieval and caching
- Automatically coordinated with business logic
- 100% pure Java
- Local, remote CORBA and EJB session bean support

Flexible Deployment

BC4J-developed services can then be deployed as CORBA Server Objects or EJB Session Beans on enterprise-scale server platforms supporting Java technology.

The same server-side business component can also be deployed without modification, as a JavaServer Pages/Servlet application or Enterprise JavaBeans component.

This flexibility, enables you to reuse the same business logic and data models to deliver applications to a variety of clients, browsers, and wireless Internet devices without needing to rewrite code.

Building BC4J XML Applications in JDeveloper

The Business Components for Java (BC4J) framework in JDeveloper9i uses XML to define the metadata that represents the declarative settings and features of the objects. Custom or complex business logic can be implemented in Java.

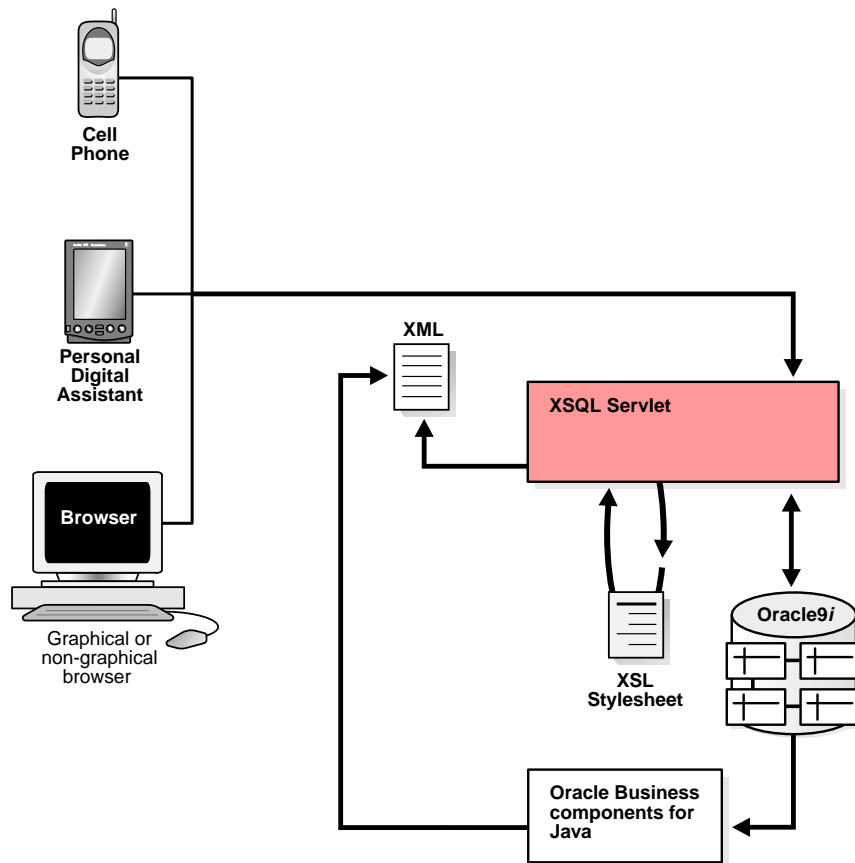
- The BC4J Tester enables you to see data in view objects as XML.
- Business rules, such as validation rules, are stored in XML rather than Java source code
- Easy customization of business applications by changing XML rather than Java source code
- Applications are easier to read and understand by abstracting the logic in XML

BC4J uses XML to Store Metadata The business components for Java framework that ships with JDeveloper uses XML to store metadata about its application components. Important information is now stored in a structured document rather than in Java source code. This makes the application easier to understand and customize.

The application is now customizable without having access to the source code.

[Figure 14-1](#) shows how BC4J is used with XSQL servlet to generate XML documents.

Figure 12–1 Using Business Components for Java (BC4J)



Business rules can be changed on site without needing access to the underlying component source code.

Building XSQL Clients with BC4J

In JDeveloper9i, you can build XSQL Pages which can integrate with BC4J application modules and thereby serve application logic from the middle tier to multiple clients. You can retrieve XML data and present it to any kind of a client device just by applying the corresponding stylesheet.

See Also:

- [Chapter 11, "Using JDeveloper to Build Oracle XML Applications"](#)
- *Oracle9i Java Developer's Guide*
- *Oracle9i Case Studies - XML Applications*

Ease of Code Generation and Management when Building XML and Java Applications

The following lists some typical JDeveloper code requirements when using the BC4J framework to build an XML application:

- A .java file and a .xml file for each entity object and each view object
- A .java file for each association object and each link object
- A .java file and a .xml file for the application module
- Double-click any of these files in the JDeveloper navigator to view the file contents.

The BC4J framework represents each Business Component that uses a combination of XML and Java code.

- **XML.** The XML code defines the metadata representing declarative settings and features of the object.
- **Java.** The Java code implements the object's behavior.

Other typical generated files are:

- Java implementation of the entity
- View XML file
- Java implementation of the view
- Application module XML file
- Java implementation of the application module

Using Metadata API

This chapter describes the following sections:

- [Introduction to Metadata API](#)
- [What is DBMS_METADATA?](#)
- [DBMS_METADATA Programmatic Interface](#)
 - [Performance Tips](#)
- [DBMS_METADATA Browsing Interface](#)
- [Metadata API Example: Retrieving DDL for Tables](#)

Introduction to Metadata API

The Metadata API provides a centralized, simple and flexible means for performing the following tasks:

- Extracting complete definitions of database objects (metadata) as either XML or creation DDL
- Transforming the metadata via industry-standard XSL-T (XML Stylesheet Transformation language).
- Generating SQL DDL to recreate the database objects

The Metadata API is available on Oracle whenever the instance is operational. It is not available on Oracle Lite.

Previous Methods Used to Extract Metadata

An object's metadata is distributed in normalized fashion across the Dictionary. In prior releases, you first had to understand how and where your object's metadata was represented in the Dictionary, then you had to issue multiple queries to extract the object's full representation. Once the metadata was extracted, you would typically perform the following tasks:

1. Transform it in some way, such as, change the object's tablespace, change a column datatype, change an object's owner, and so on.
2. Convert it to SQL DDL text for execution on the source or some other database.

In prior releases, there was no assistance for either of these steps.

Metadata API Components

Underlying the Metadata API is an object model of the Oracle Dictionary comprised of a series of User-Defined Types (UDTs) and corresponding object views. The UDTs provide the aggregation of each object class's metadata and the object views map the UDTs' attributes onto the appropriate base relational tables in the Dictionary. The Metadata API generates queries against these object views to retrieve aggregated database object definitions.

The results from these queries are converted into XML documents by the XML / SQL utility (also new in Oracle). When the caller requests DDL output, the Metadata API uses the Oracle server's integral XML Parser and XSL Processor to convert the XML documents into creation DDL.

Metadata API Features

The Metadata API has the following features:

- Provides a powerful PL/SQL interface for detailed programmatic control or casual browsing.
- Supports retrieval of complete, aggregated database object definitions for the following classes of objects:
 - All types of tables (including relational, object, index-organized, nested, temporary and partitioned)
 - Indexes (including functional and domain indexes)
 - User-defined types
 - Procedures, functions, and packages
 - Operators
 - Indextypes
 - Relational and object views
 - Triggers
 - Synonyms
 - Grants (object and system privilege grants)
 - Outlines
- Provides *only* complete representations of objects.

Note: Subsetting of object attributes is not supported in this release except through XSLT transformation.

- Provides database object metadata in an XML format that is easily transformable via XSL-T by downstream processes.
- Provides complete Oracle-specific creation DDL for all supported objects.
- Provides flexible object selection. Can return multiple objects per query.
- Supports daisy-chained transforms where the output of the first becomes the input to the second and so on.
- DDL output can be customized via object type-specific transform parameters.

Internet Computing

Metadata API uses two internet standards, XML and XSLT, for encoding and transforming object metadata. Use of an industry-standard format for metadata encoding (rather than a proprietary format) allows you to use standard tools to parse and transform the output.

There is currently no industry-standard XML model for database metadata, so Metadata API uses one optimized for generating Oracle DDL. Document element names are derived directly from attributes of the UDTs in the Oracle Dictionary model. As standard models emerge, Metadata API will support the ability to plug them in. Older documents can be converted to alternate models with XSLT.

With n-tier Internet Computing, it is natural for Metadata API to be bound to the server, close to the metadata. Hence, the Metadata API's implementation chose PL/SQL, which is callable from any other language including Java.

What is DBMS_METADATA?

DBMS_METADATA is the PL/SQL package that implements Metadata API. It allows callers to retrieve metadata from the database Dictionary. It provides a flexible and extensible means for object selection. You can use DBMS_METADATA to extract database object metadata in XML and DDL.

DBMS_METADATA has two types of interface:

- **Programmatic interface** for fine-grained, detailed control:
 - The following routines are provided and explained later: OPEN, SET_FILTER, SET_PARSE_ITEM, SET_COUNT, ADD_TRANSFORM, SET_TRANSFORM_PARAM, FETCH_XXX, CLOSE
 - Metadata is expressed as XML. This allows industry-standard metadata transformations using XSLT.
 - You can ask DBMS_METADATA to return metadata as DDL. The API uses XSL scripts internally to transparently perform the conversion.
 - You can simply invoke an XSL script, using either Oracle's XML parser or some third-party tool, to do an off-line conversion of the XML representation.
- **Browsing interface** for casual use within SQL clients such as SQL*Plus:
 - GET_DDL, GET_XML. For example, the following query will show the DDL for all tables in the current user's schema:

```
SELECT dbms_metadata.get_ddl('TABLE', table_name) FROM user_tables;
```

- Session-level “sticky” transform parameters

DBMS_METADATA and Security

The object views of the Oracle metadata model implement security. Non-privileged users can see the metadata of just their own objects. SYS and those users with SELECT_CATALOG_ROLE can see all objects. Non-privileged users can also retrieve object and system privileges granted to them or by them to others. This also includes privileges granted to PUBLIC.

If callers request objects they are not privileged to retrieve, no exception is raised; the object is simply not retrieved.

Note: If non-privileged users are granted some form of access to an object in someone else’s schema, they will be able to retrieve the grant specification through the Metadata API, but not the object’s actual metadata.

Note: The types and public interface defined by the Metadata API can be found in:

```
$ORACLE_HOME/rdbms/admin/dbmsmeta.sql
```

See Also: *Oracle9i Supplied PL/SQL Packages Reference*

DBMS_METADATA Programmatic Interface

Table 13–1 lists the DBMS_METADATA programmatic interface procedures.

Table 13–1 DBMS_METADATA Procedures: Programmatic Interface

PL/SQL Procedure	Syntax	Description
DBMS_METADATA.OPEN()	FUNCTION open (object_type IN VARCHAR2, version IN VARCHAR2 DEFAULT 'COMPATIBLE', model IN VARCHAR2 DEFAULT 'ORACLE') RETURN NUMBER;	Specifies type of object to be retrieved, version of its metadata, and object model. Return value is an opaque context handle for the set of objects to be used in subsequent calls.
DBMS_METADATA.SET_FILTER()	PROCEDURE set_filter (handle IN NUMBER, name IN VARCHAR2, value IN VARCHAR2); PROCEDURE set_filter (handle IN NUMBER, name IN VARCHAR2, value IN BOOLEAN DEFAULT TRUE);	Specifies restrictions on objects to be retrieved, such as, object name or schema. Allows specification of base object(s) for dependent objects such as INDEXes and TRIGGERs.
DBMS_METADATA.SET_COUNT()	PROCEDURE set_count (handle IN NUMBER, value IN NUMBER);	Specifies number of objects to be retrieved in a single FETCH_xxx call. By default, each call to FETCH_xxx returns one object.
DBMS_METADATA.GET_QUERY()	FUNCTION get_query (handle IN NUMBER) RETURN VARCHAR2;	Returns text of query (or queries) used by FETCH_xxx. Provided to assist in debugging.
DBMS_METADATA.SET_PARSE_ITEM()	PROCEDURE set_parse_item (handle IN NUMBER, name IN VARCHAR2);	Enables output parsing and specifies an object attribute to be parsed and returned. This frees the caller from having to parse SQL DDL for key attributes.

Table 13–1 DBMS_METADATA Procedures: Programmatic Interface (Cont.)

PL/SQL Procedure	Syntax	Description
DBMS_METADATA.ADD_TRANSFORM()	<pre> FUNCTION add_transform (handle IN NUMBER, name IN VARCHAR2 encoding IN VARCHAR2 DEFAULT NULL)) RETURN NUMBER;</pre>	<p>Specifies a transform that <code>FETCH_xxx</code> applies to the XML representation of retrieved objects. You can add more than one transform. By default (with no transforms added), objects are returned as XML documents. Call <code>ADD_TRANSFORM</code> to specify an XSLT script to transform the returned documents. If 'DDL' is specified, the objects' creation DDL is returned from subsequent <code>FETCH_xxx</code> calls. <code>ADD_TRANSFORM</code> returns an opaque transform handle different from that returned by <code>OPEN</code>.</p> <p>Specify <i>encoding</i> if:</p> <ul style="list-style-type: none"> ■ The XSL stylesheet pointed to by an external URL is encoded in a character set that is not a subset of UTF-8, or ■ The XSL stylesheet pointed to by a DB-internal URL is encoded in a character set that is not a subset of the database's character set.

Table 13–1 DBMS_METADATA Procedures: Programmatic Interface (Cont.)

PL/SQL Procedure	Syntax	Description
DBMS_METADATA.SET_TRANSFORM_PARAM()	PROCEDURE set_transform_param (transform_handle IN NUMBER, name IN VARCHAR2, value IN VARCHAR2); PROCEDURE set_transform_param (transform_handle IN NUMBER, name IN VARCHAR2, value IN BOOLEAN DEFAULT TRUE);	Specifies parameters to the XSLT stylesheet identified by <i>transform_handle</i> returned from ADD_TRANSFORM. For the DDL transform, these parameters alter the form of the DDL. For example, constraints may be requested as column constraints or ALTER TABLE statements.
DBMS_METADATA.FETCH_xxx()	FUNCTION fetch_xml (handle IN NUMBER) RETURN XMLType; FUNCTION fetch_ddl (handle IN NUMBER) RETURN sys.ku\$ddls; FUNCTION fetch_clob (handle IN NUMBER) RETURN CLOB; PROCEDURE fetch_clob (handle IN NUMBER, doc IN OUT NOCOPY CLOB);	The FETCH_xxx routines return metadata for objects meeting the criteria established by OPEN, SET_FILTER, SET_COUNT, ADD_TRANSFORM... FETCH_XML and FETCH_DDL return the metadata as XML and SQL DDL, respectively. The FETCH_CLOB routines return either XML or DDL as denoted by the transforms specified. The types used by these routines are described in <i>Oracle9i Supplied PL/SQL Packages Reference</i> .
DBMS_METADATA.CLOSE()	PROCEDURE close (handle IN NUMBER);	Invalidates the handle returned by OPEN and cleans up associated state.

See Also:

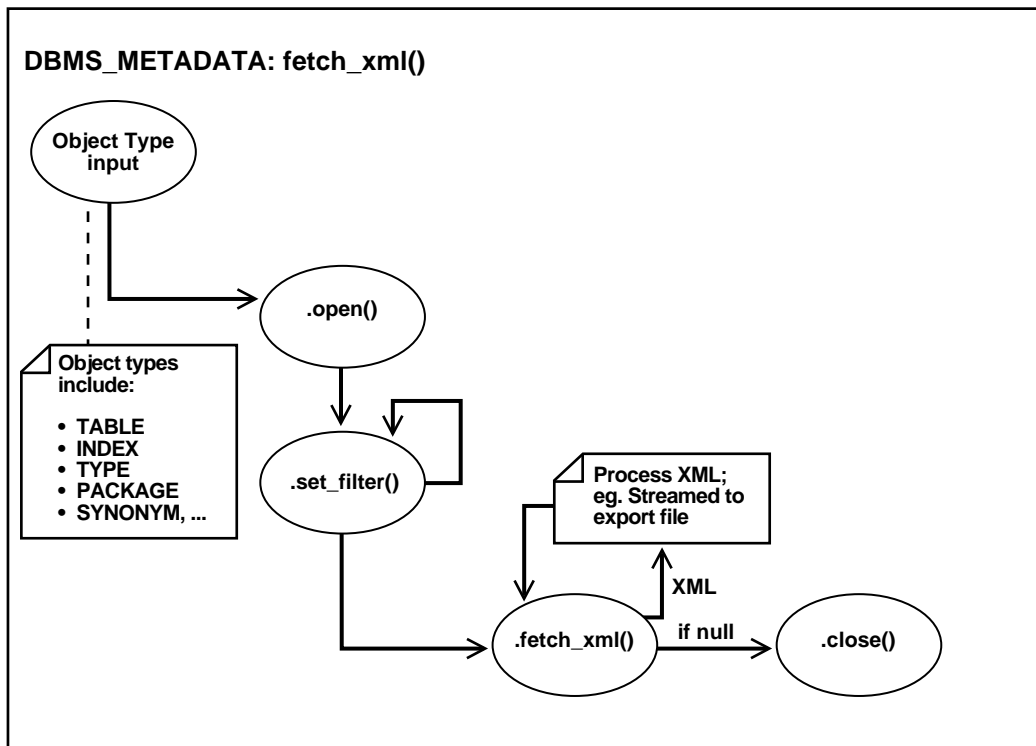
- *Oracle9i Supplied PL/SQL Packages Reference*
- [Chapter 5, "Database Support for XML"](#) for the specification of XMLType.

DBMS_METADATA.FETCH_XML

Figure 13-1 illustrates `DBMS_METADATA.FETCH_XML()` usage:

1. Open the object type using `DBMS_METADATA.OPEN()`.
2. Specify which objects to retrieve using `DBMS_METADATA.SET_FILTER()`.
3. Fetch each qualifying object's metadata as an XML document using `DBMS_METADATA.FETCH_XML()`.
4. If the result of this operation is `NULL`, then `DBMS_METADATA.CLOSE()`.

Figure 13-1 Using DBMS_METADATA.FETCH_XML()



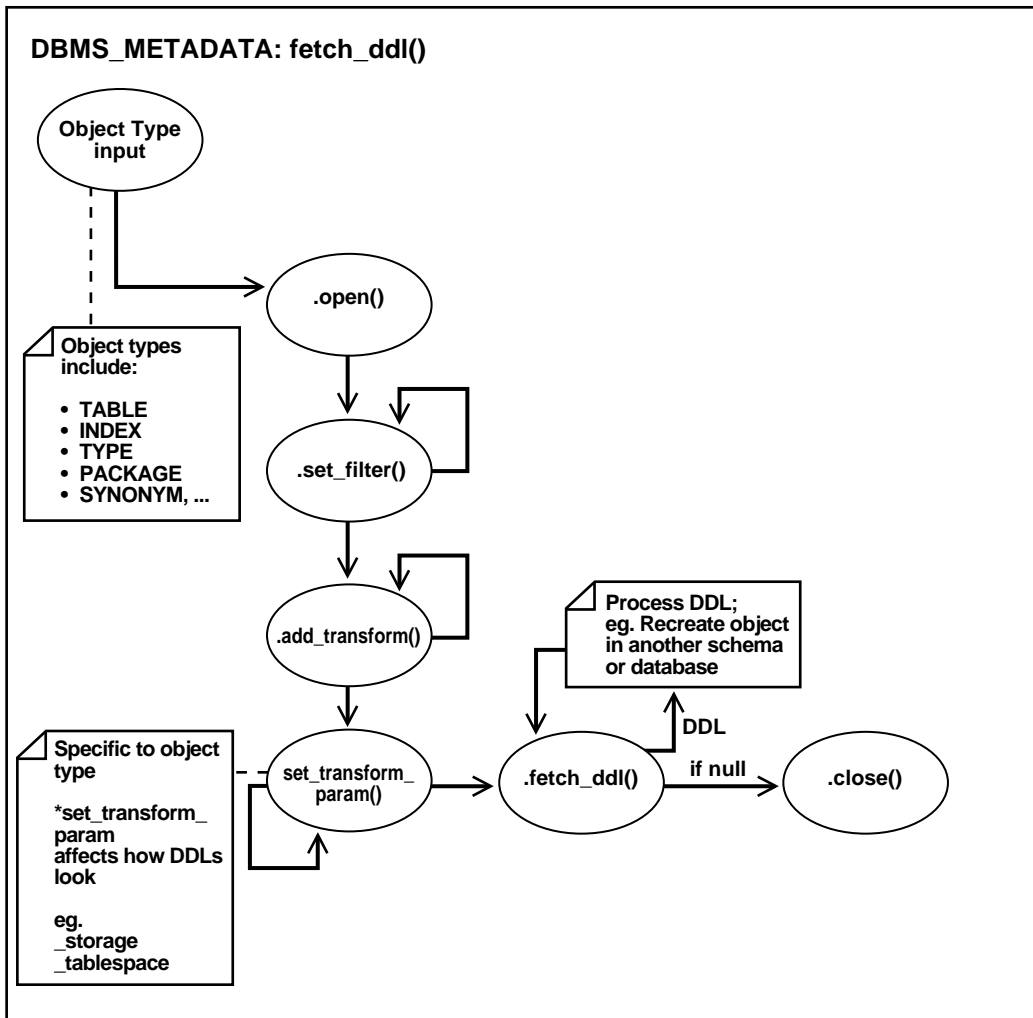
DBMS_METADATA.FETCH_DDL()

Figure 13-2 illustrates DBMS_METADATA.FETCH_DDL() usage:

1. Open the object type using `DBMS_METADATA.OPEN()`.
2. Specify which objects to retrieve using `DBMS_METADATA.SET_FILTER()`.
3. Specify what transforms are to be invoked on the output:
 - `DBMS_METADATA.ADD_TRANSFORM()` to add a transform. The last transform added must be the "DDL" transform.

4. `DBMS_METADATA.SET_TRANSFORM_PARAM()`. This allows you to customize the DDL; for example, to exclude storage clauses on table definitions. Transform parameters are specific to the object type chosen.
5. Fetch the DDL using `DBMS_METADATA.FETCH_DDL()`.
6. If the result of this operation is `NULL`, then `DBMS_METADATA.CLOSE()`.

Figure 13-2 Using DBMS_METADATA.FETCH_DDL()



Performance Tips

This section describes how to enhance performance when using the programmatic interface of Metadata API.

1. Fetch all of one type of object before fetching the next. For example, if you are retrieving the definitions of all objects in your schema, first fetch all the tables, then all the indexes, then all the triggers, and so on. This will be much faster than nesting OPEN contexts; that is, fetch one table then all of its indexes, grants and triggers, then the next table and all of its indexes, grants and triggers, and so on. The example at the end of this chapter actually reflects this second, less efficient means, but its purpose is to demonstrate most of the programmatic calls which are best shown by this method.
2. Use the SET_COUNT procedure to retrieve more than one object at a time. This minimizes server round trips as well as eliminates many redundant function calls.
3. Use the procedure rather than function form of FETCH_CLOB. The procedure form returns the output CLOB by reference via the IN OUT NOCOPY specifier. The function form returns the output CLOB by value requiring an extra LOB copy.
4. When writing a PL/SQL package that calls Metadata API, declare LOB variables and objects that contain LOBs (such as SYS.KU\$_DDL) at package scope rather than within individual functions. This eliminates the creation and deletion of LOB duration structures upon function entrance and exit which are very expensive operations.

See Also: *Oracle9i Application Developer's Guide - Large Objects (LOBs)*

DBMS_METADATA Browsing Interface

The DBMS_METADATA browsing interface is provided by the GET_XML and GET_DDL functions.

[Table 13-2](#) lists the browsing APIs, their syntax, and a brief description.

Table 13–2 DBMS_METADATA Procedures: Browsing Interface

PL/SQL Procedure Name	Syntax	Description
DBMS_METADATA.GET_XXX()	<pre> FUNCTION get_xml (object_type IN VARCHAR2, name IN VARCHAR2, schema IN VARCHAR2 DEFAULT NULL, version IN VARCHAR2 DEFAULT 'COMPATIBLE', model IN VARCHAR2 DEFAULT 'ORACLE', transform IN VARCHAR2 DEFAULT NULL) RETURN CLOB; FUNCTION get_ddl (object_type IN VARCHAR2, name IN VARCHAR2, schema IN VARCHAR2 DEFAULT NULL, version IN VARCHAR2 DEFAULT 'COMPATIBLE', model IN VARCHAR2 DEFAULT 'ORACLE', transform IN VARCHAR2 DEFAULT 'DDL') RETURN CLOB; </pre>	<p>Provides a way to return metadata for a single object. Each GET_XXX call is comprised of an OPEN, one or two SET_FILTER calls, optionally an ADD_TRANSFORM, a FETCH_XXX and a CLOSE.</p> <p>The <i>object_type</i> parameter has the same semantics as in OPEN. <i>schema</i> and <i>name</i> are used for filtering.</p> <p>If a transform is specified, session-level transform flags are inherited.</p>

Example

The following SQL*Plus command will display the creation DDL for all tables in the current user's schema:

```
SQL> SELECT dbms_metadata.get_ddl('TABLE', table_name) FROM user_tables;
```

Metadata API Example: Retrieving DDL for Tables

Here is a detailed Metadata API programming example, `PAYROLL_DEMO`, that retrieves the DDL for all tables in the `MDDEMO` schema that start with 'PAYROLL'. It then fetches the DDL for grants, indexes and triggers defined on those tables. This script can be found in the file `rdbms/demo/mddemo.sql` in your Oracle home directory.

mddemo.sql

```
-- This script demonstrates how to use the Metadata API. It first
-- establishes a schema (MDDEMO) and some payroll users, then creates three
-- payroll-like tables within it along with associated indexes, triggers
-- and grants.

-- It then creates a package PAYROLL_DEMO that shows common usage of the
-- Metadata API. The procedure GET_PAYROLL_TABLES retrieves the DDL for the
-- two tables in this schema that start with 'PAYROLL' then for each one,
-- retrieves the DDL for its associate dependent objects; indexes, grants
-- and triggers. All the DDL is written to a table named "MDDEMO"."DDL".

-- First, Install the demo. cd to rdbms/demo:
-- > sqlplus system/manager
-- SQL> @mddemo

-- Then, run it.
-- > sqlplus mddemo/mddemo
-- SQL> set long 40000
-- SQL> set pages 0
-- SQL> call payroll_demo.get_payroll_tables();
-- SQL> select ddl from DDL order by seqno;

Rem Set up schema for demo pkg. PAYROLL_DEMO.

connect system/manager
drop user mddemo cascade;
drop user mddemo_clerk cascade;
drop user mddemo_mgr cascade;

create user mddemo identified by mddemo;
GRANT resource, connect, create session
      , create table
      , create procedure
      , create sequence
```

```
        , create trigger
        , create view
        , create synonym
        , alter session
TO mddemo;

create user mddemo_clerk identified by clerk;
create user mddemo_mgr identified by mgr;

connect mddemo/mddemo

Rem Create some payroll-like tables...

create table payroll_emps
( lastname varchar2(60) not null,
  firstname varchar2(20) not null,
  mi varchar2(2),
  suffix varchar2(10),
  DOB date not null,
  badge_no number(6) primary key,
  exempt varchar(1) not null,
  salary number (9,2),
  hourly_rate number (7,2)
)
/
create table payroll_timecards
  badge_no number(6) references payroll_emps (badge_no),
  week number(2),
  job_id number(5),
  hours_worked number(4,2)
)
/
-- This is a dummy table used only to show that tables NOT starting with
-- 'PAYROLL' are NOT retrieved by payroll_demo.get_payroll_tables

create table audit_trail
(action_time DATE,
lastname VARCHAR2(60),
action LONG
)
/

Rem Then, create some grants...

grant update (salary,hourly_rate) on payroll_emps to mddemo_clerk;
```



```
grant ALL on payroll_emps to mddemo_mgr with grant option;

grant insert,update on payroll_timecards to mddemo_clerk;
grant ALL on payroll_timecards to mddemo_mgr with grant option;

Rem Then, create some indexes...

create index i_payroll_emps_name on payroll_emps(lastname);
create index i_payroll_emps_dob on payroll_emps(DOB);

create index i_payroll_timecards_badge on payroll_timecards(badge_no);

Rem Then, create some triggers (and required procedure)...

create or replace procedure check_sal( salary in number) as
begin
    return; -- Fairly loose security here...
end;
/

create or replace trigger salary_trigger before insert or update of salary on
payroll_emps
for each row when (new.salary > 150000)
call check_sal(:new.salary)
/

create or replace trigger hourly_trigger before update of hourly_rate on
payroll_emps
for each row
begin :new.hourly_rate:=:old.hourly_rate;end;
/

--
-- Set up a table to hold the generated DDL
--
CREATE TABLE ddl (ddl CLOB, seqno NUMBER);

Rem Finally, create the PAYROLL_DEMO package itself.

CREATE OR REPLACE PACKAGE payroll_demo AS

    PROCEDURE get_payroll_tables;
END;
/
CREATE OR REPLACE PACKAGE BODY payroll_demo AS
```

```

-- GET_PAYROLL_TABLES: Fetch DDL for payroll tables and their dependent objects

PROCEDURE get_payroll_tables IS

tableOpenHandle NUMBER;
depObjOpenHandle NUMBER;
tableTransHandle NUMBER;
indexTransHandle NUMBER;
schemaName VARCHAR2(30);
tableName VARCHAR2(30);
tableDDLs sys.ku$_ddl;
tableDDL sys.ku$_ddl;
parsedItems sys.ku$_parsed_items;
depObjDDL CLOB;
seqNo NUMBER := 1;

TYPE obj_array_t IS VARRAY(3) OF VARCHAR2(30);

-- Load this array with the dependent object classes to be retrieved...
obj_array obj_array_t := obj_array_t('OBJECT_GRANT', 'INDEX', 'TRIGGER');

BEGIN

-- Open a handle for tables in the current schema.
tableOpenHandle := dbms_metadata.open('TABLE');

-- Tell mdAPI to retrieve one table at a time. This call is not actually
-- necessary since 1 is the default... just showing the call.
dbms_metadata.set_count(tableOpenHandle, 1);

-- Retrieve tables whose name starts with 'PAYROLL'. When the filter is
-- 'NAME_EXPR', the filter value string must include the SQL operator. This
-- gives the caller flexibility to use LIKE, IN, NOT IN, subqueries, etc.
dbms_metadata.set_filter(tableOpenHandle, 'NAME_EXPR', 'LIKE ''PAYROLL%'');

-- There are no index-organized tables in the MDDEMO schema, so tell the API.
-- This eliminates one of the views it'll need to look in.
dbms_metadata.set_filter(tableOpenHandle, 'IOT', FALSE);

-- Tell the mdAPI to parse out each table's schema and name separately so we
-- can use them to set up the calls to retrieve its dependent objects.
dbms_metadata.set_parse_item(tableOpenHandle, 'SCHEMA');
dbms_metadata.set_parse_item(tableOpenHandle, 'NAME');

```

```

-- Add the DDL transform so we get SQL creation DDL
tableTransHandle := dbms_metadata.add_transform(tableOpenHandle, 'DDL');

-- Tell the XSL stylesheet we don't want physical storage information (storage,
-- tablespace, etc), and that we want a SQL terminator on each DDL. Notice that
-- these calls use the transform handle, not the open handle.
dbms_metadata.set_transform_param(tableTransHandle,
    'SEGMENT_ATTRIBUTES', FALSE);
dbms_metadata.set_transform_param(tableTransHandle,
    'SQLTERMINATOR', TRUE);

-- Ready to start fetching tables. We use the FETCH_DDL interface (rather than
-- FETCH_XML or FETCH_CLOB). This interface returns a SYS.KU$_DDL; a table of
-- SYS.KU$_DDL objects. This is a table because some object types return
-- multiple DDL statements (like types / pkgs which have create header and
-- body statements). Each KU$_DDL has a CLOB containing the 'CREATE foo'
-- statement plus a nested table of the parse items specified. In our case,
-- we asked for two parse items; Schema and Name. (NOTE: See admin/dbmsmeta.sql
-- for a more detailed description of these types)

LOOP
    tableDDLs := dbms_metadata.fetch_ddl(tableOpenHandle);
    EXIT WHEN tableDDLs IS NULL; -- Get out when no more payroll tables

-- In our case, we know there is only one row in tableDDLs (a KU$_DDLs tbl obj)
-- for the current table. Sometimes tables have multiple DDL statements;
-- eg, if constraints are applied as ALTER TABLE statements, but we didn't ask
-- for that option. So, rather than writing code to loop through tableDDLs,
-- we'll just work with the 1st row.
--
-- First, write the CREATE TABLE text to our output table then retrieve the
-- parsed schema and table names.
    tableDDL := tableDDLs(1);
    INSERT INTO ddl VALUES(tableDDL.ddltext, seqNo);
    seqNo := seqNo + 1;
    parsedItems := tableDDL.parsedItems;

-- Must check the name of the returned parse items as ordering isn't guaranteed
FOR i IN 1..2 LOOP
    IF parsedItems(i).item = 'SCHEMA'
    THEN
        schemaName := parsedItems(i).value;
    ELSE
        tableName := parsedItems(i).value;
    END IF;

```

```

        END LOOP;

-- Now, we want to retrieve all the dependent objects defined on the current
-- table: indexes, triggers and grants. Since all 'dependent' object types
-- have BASE_OBJECT_NAME and BASE_OBJECT_SCHEMA in common as filter criteria,
-- we'll set up a loop to get all objects of the 3 types, just changing the
-- OPEN context in each pass through the loop. Transform parameters are
-- different for each object type, so we'll only use one that's common to all;
-- SQLTERMINATOR.

FOR i IN 1..3 LOOP
    depObjOpenHandle := dbms_metadata.open(obj_array(i));
    dbms_metadata.set_filter(depObjOpenHandle, 'BASE_OBJECT_SCHEMA',
        schemaName);
    dbms_metadata.set_filter(depObjOpenHandle, 'BASE_OBJECT_NAME', tableName);

-- Add the DDL transform and say we want a SQL terminator
    indexTransHandle := dbms_metadata.add_transform(depObjOpenHandle, 'DDL');
    dbms_metadata.set_transform_param(indexTransHandle,
        'SQLTERMINATOR', TRUE);

-- Retrieve dependent object DDLs as CLOBs and write them to table DDL.
    LOOP
        depObjDDL := dbms_metadata.fetch_clob(depObjOpenHandle);
        EXIT WHEN depObjDDL IS NULL;
        INSERT INTO ddl VALUES(depObjDDL, seqNo);
        seqNo := seqNo + 1;
    END LOOP;

-- Free resources allocated for current dependent object stream.
    dbms_metadata.close(depObjOpenHandle);

    END LOOP; -- End of fetch dependent objects loop

END LOOP; -- End of fetch table loop

-- Free resources allocated for table stream and close output file.
dbms_metadata.close(tableOpenHandle);
RETURN;

END; -- of procedure get_payroll_tables

END payroll_demo;
/

```

PAYROLL_DEMO Output

This is the output obtained from executing procedure, `mddemo.payroll_demo.get_payroll_tables`. The output is obtained by executing the following query as user `mddemo`:

```
SQL> SELECT ddl FROM ddl ORDER BY seqno;

CREATE TABLE "MDDemo"."PAYROLL_EMPS"
  (
    "LASTNAME" VARCHAR2(60) NOT NULL ENABLE,
    "FIRSTNAME" VARCHAR2(20) NOT NULL ENABLE,
    "MI" VARCHAR2(2),
    "SUFFIX" VARCHAR2(10),
    "DOB" DATE NOT NULL ENABLE,
    "BADGE_NO" NUMBER(6,0),
    "EXEMPT" VARCHAR2(1) NOT NULL ENABLE,
    "SALARY" NUMBER(9,2),
    "HOURLY_RATE" NUMBER(7,2),
    PRIMARY KEY ("BADGE_NO") ENABLE
  ) ;

GRANT UPDATE ("SALARY") ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_CLERK";
GRANT UPDATE ("HOURLY_RATE") ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_CLERK";
GRANT ALTER ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT DELETE ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT INDEX ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT INSERT ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT SELECT ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT UPDATE ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT REFERENCES ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT ON COMMIT REFRESH ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT
OPTION;
GRANT QUERY REWRITE ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTI
ON;

CREATE INDEX "MDDemo"."I_PAYROLL_EMPS_DOB" ON "MDDemo"."PAYROLL_EMPS" ("DOB")
PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;

CREATE INDEX "MDDemo"."I_PAYROLL_EMPS_NAME" ON "MDDemo"."PAYROLL_EMPS" ("LASTIN
AME")
PCTFREE 10 INITRANS 2 MAXTRANS 255
```

```

STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;

CREATE OR REPLACE TRIGGER hourly_trigger before update of hourly_rate on payro
ll_emps
for each row
begin :new.hourly_rate:=:old.hourly_rate;end;
/
ALTER TRIGGER "MDEMO"."HOURLY_TRIGGER" ENABLE;

CREATE OR REPLACE TRIGGER salary_trigger before insert or update of salary on
payroll_emps
for each row WHEN (new.salary > 150000) CALL check_sal(:new.salary)
/
ALTER TRIGGER "MDEMO"."SALARY_TRIGGER" ENABLE;

CREATE TABLE "MDEMO"."PAYROLL_TIMECARDS"
(
  "BADGE_NO" NUMBER(6,0),
  "WEEK" NUMBER(2,0),
  "JOB_ID" NUMBER(5,0),
  "HOURS_WORKED" NUMBER(4,2),
  FOREIGN KEY ("BADGE_NO")
  REFERENCES "MDEMO"."PAYROLL_EMPS" ("BADGE_NO") ENABLE
) ;

GRANT INSERT ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_CLERK";
GRANT UPDATE ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_CLERK";
GRANT ALTER ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_MGR" WITH GRANT OPTION;
GRANT DELETE ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_MGR" WITH GRANT OPTION
;
GRANT INDEX ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_MGR" WITH GRANT OPTION;
GRANT INSERT ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_MGR" WITH GRANT OPTION
;
GRANT SELECT ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_MGR" WITH GRANT OPTION
;
GRANT UPDATE ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_MGR" WITH GRANT OPTION
;
GRANT REFERENCES ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_MGR" WITH GRANT OP
TION;
GRANT ON COMMIT REFRESH ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_MGR" WITH G
RANT OPTION;
GRANT QUERY REWRITE ON "MDEMO"."PAYROLL_TIMECARDS" TO "MDEMO_MGR" WITH GRANT
OPTION;

```

```
CREATE INDEX "MDDEMO"."I_PAYROLL_TIMECARDS_BADGE" ON "MDDEMO"."PAYROLL_TIMECARDS" ("BADGE_NO")
PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;
```

OracleAS Reports Services and XML

This chapter contains the following sections:

- [Introducing OracleAS Reports Services and XML](#)
- [Creating XML Output 'On the Fly' Using OracleAS Reports Services](#)
- [Customizing Report Definitions at Runtime](#)
- [Performing Batch Report Modifications by Applying XML Report Definitions](#)
- [Creating Report Definitions in XML](#)
- [Using XML as a Datasource](#)
- [Reports Case Studies](#)
- [Frequently Asked Questions: Reports and XML](#)

Introducing OracleAS Reports Services and XML

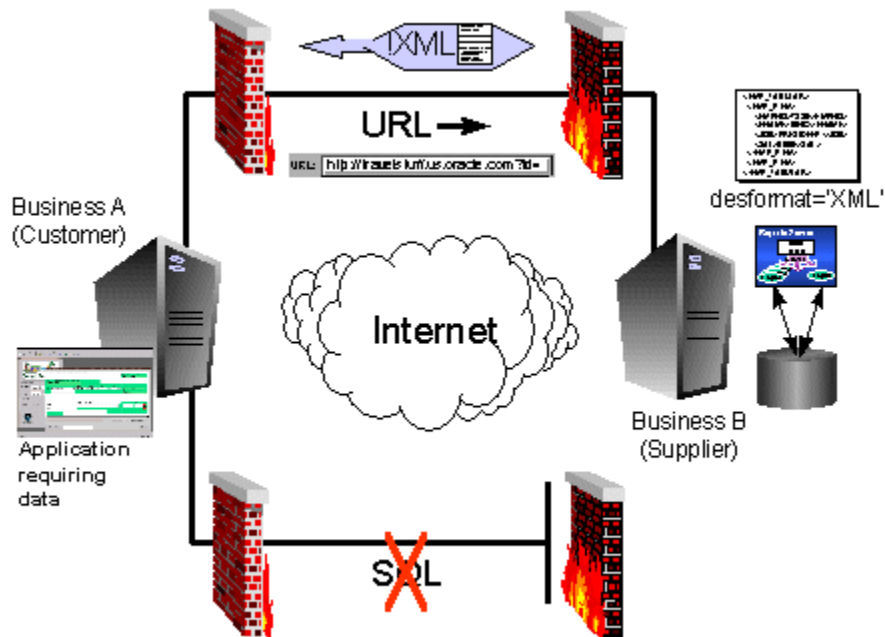
OracleAS Reports Services provides the following XML support:

- Reports can be output and customized as XML [Reports 6i and higher].
 - Create XML output 'on the fly', using OracleAS Reports Services. See ["Creating XML Output "On the Fly" Using OracleAS Reports Services"](#) on page 14-4.
 - Customize Report Definitions at runtime. See ["Customizing Report Definitions at Runtime"](#) on page 14-6.
 - Run Batch report modifications using XML Report Definitions. See ["Performing Batch Report Modifications by Applying XML Report Definitions"](#) on page 14-12.
- Read XML data source from Reports [Reports 9i and higher]. See ["Using XML as a Datasource"](#) on page 14-17.

B2B Data Exchange: Why Use XML in Reports?

[Figure 14-1](#) shows the sharing of information with partners and how XML can be used to help send data in a more timely manner. With OracleAS Reports Server, you can automatically generate XML files. A URL is all that is required to invoke an Oracle Report on the Web. By defining a Report module to query the data, suppliers can stream information back to the calling eCommerce application in real time.

Figure 14–1 Why Use XML in OracleAS Reports Services?



See Also:

- *Oracle9i Case Studies - XML Applications*, under the chapter, "Customizing Discoverer4i (9i) Viewer with XSL"
- *"Oracle Reports Developer Release 6i: Building Reports"* or later.
- *"Oracle Reports Developer Release 6i: Publishing Reports"* or later.
- *"Oracle Reports Developer Release 6i Reference Manual"* or later.
- Online help, accessed in several ways, for example from your desktop, go to START > "Oracle Forms and Reports Doc".

What's Needed to Run OracleAS Reports Services

Oracle Reports, or OracleAS Reports Service as it is called now, is part of Oracle Application Server. It seamlessly integrates in the environment and the other services provided by the product.

See Also: <http://otn.oracle.com/products/reports/>

Creating XML Output "On the Fly" Using OracleAS Reports Services

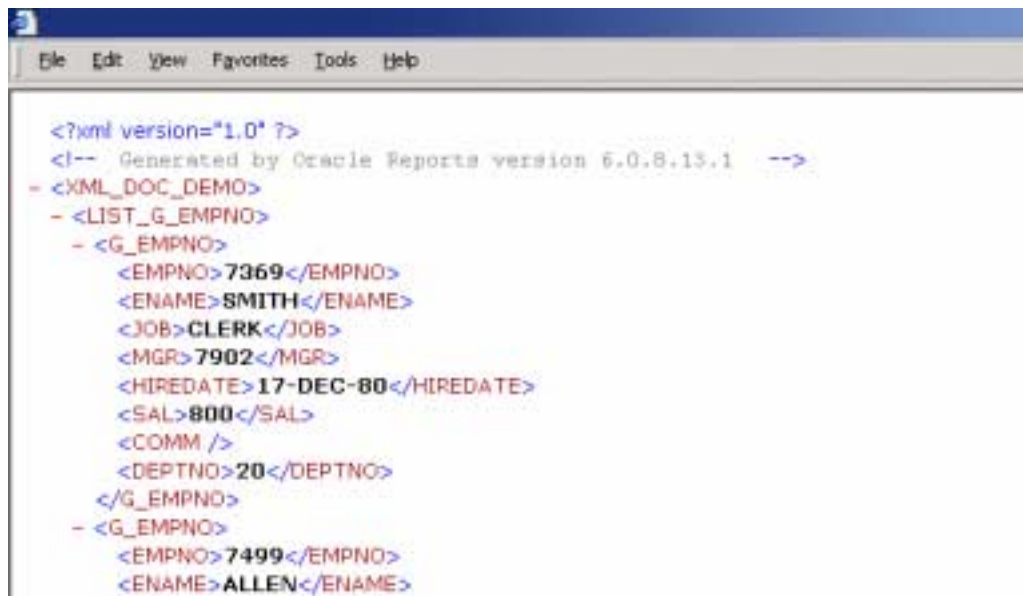
The data model drives the structure of XML output for a report. XML output is not dependent on the visual layout.

- You can change the structure of a report's XML output by editing the XML properties available in the Property Palette for data model objects.
- Report Builder does not currently support presentation information (color, font, physical layout, etc.) in XML output. You may provide your own presentation information using XSL, which may be specified in the prolog.
- From the command line, you can use the argument `DESFORMAT=XML` to generate XML output.

See Also: ■ *"Oracle Reports Developer Release 6i Reference Manual"*, for more information on `DESFORMAT`.

XML as a Data InterChange Format

You may need to send a report to a B2B partner in XML. [Figure 14-2](#) shows an example of a report in XML. This is obtained by changing one parameter that tells Reports Server to output the report in XML instead of HTML.

Figure 14-2 Example Report Shown in XML

```
<?xml version="1.0" ?>
<!-- Generated by Oracle Reports version 6.0.8.13.1 -->
- <XML_DOC_DEMO>
- <LIST_G_EMPNO>
  - <G_EMPNO>
    <EMPNO>7369</EMPNO>
    <ENAME>SMITH</ENAME>
    <JOB>CLERK</JOB>
    <MGR>7902</MGR>
    <HIREDATE>17-DEC-80</HIREDATE>
    <SAL>800</SAL>
    <COMM />
    <DEPTNO>20</DEPTNO>
  </G_EMPNO>
  - <G_EMPNO>
    <EMPNO>7499</EMPNO>
    <ENAME>ALLEN</ENAME>
```

Formatting XML Output Using XSL Stylesheets

Figure 14-3 shows the results of applying an XSL stylesheet to the same XML report shown in Figure 14-2. In this case, the example report of data is needed in HTML.

Figure 14–3 Example Report After Applying an XSL Stylesheet


<i>Empno</i>	<i>Ename</i>	<i>Job</i>	<i>Mgr</i>	<i>Hiredate</i>	<i>Sal</i>
7369	SMITH	CLERK	7902	17-DEC-80	800
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600
7521	WARD	SALESMAN	7698	22-FEB-81	1250
7566	JONES	MANAGER	7839	02-APR-81	2975
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250
7698	BLAKE	MANAGER	7839	01-MAY-81	2850
7782	CLARK	MANAGER	7839	09-JUN-81	2450
7788	SCOTT	ANALYST	7566	09-DEC-82	3000
7839	KING	PRESIDENT		17-NOV-81	5000

Customizing Report Definitions at Runtime

OracleAS Reports Services allows modifications to be externalized into a separate 'customization' file, rather than having to create unique versions of each report. Report outputs can be customized for specific users or groups without changing the original report definition.

A customization file is a report definition that is applied to an *existing* report (.RDF or .XML). It can change certain characteristics of existing report objects, such as the field's date format mask or background color. A customization file can also be used to add entirely new objects to another report.

See Also: ["Using XML as a Datasource"](#) on page 14-17

Applying an XML Customization

To apply an XML report definition to an .RDF or .XML file at runtime, use either of the following:

- **Runtime options.** The CUSTOMIZE command line argument. CUSTOMIZE can be used with RWCLI60, RWRUN60, RWBLD60, RWCON60, and URL report requests.
- **Built-In Options.** The SRW.APPLY_DEFINITION built-in (a supplied PL/SQL package)

Applying One Customization File

The following command line sends a job request to OracleAS Reports Services that applies an XML report definition, `emp.xml`, to an .RDF file, `emp.rdf`:

```
rwcli60 report=emp.rdf customize=e:\myreports\emp.xml
      userid=username/password@mydb destype=file desname=emp.pdf desformat=PDF
      server=repserver
```

Reports Runtime. If you were using OracleAS Reports Services Runtime, then the equivalent command line would be:

```
rwr60 userid=username/password@mydb report=emp.rdf
      customize=e:\myreports\emp.xml destype=file desname=emp.pdf
      desformat=PDF
```

When testing your XML report definition, it is sometimes useful to run your report requests with additional arguments to create a trace file. For example:

```
tracefile=emp.log tracemode=trace_replace traceopt=trace_app
```

The trace file provides a detailed listing of the creation and formatting of the report objects.

Applying Multiple XML Customization Files

You can apply multiple XML report definitions to a report at runtime by providing a list with the `CUSTOMIZE` command line argument. The following command line sends a job request to OracleAS Reports Services that applies two XML report definitions, `emp0.xml` and `emp1.xml`, to an .RDF file, `emp.rdf`:

```
rwcli60 report=emp.rdf
      customize="(e:\corp\myreports\emp0.xml,
      e:\corp\myreports\emp1.xml)"
      userid=username/password@mydb destype=file desname=emp.pdf desformat=PDF
      server=repserver
```

Reports Runtime. If you are using OracleAS Reports Services Runtime, then the equivalent command line is:

```
rwr60 report=emp.rdf
      customize="(e:\corp\myreports\emp0.xml,
      e:\corp\myreports\emp1.xml)"
      userid=username/password@mydb destype=file desname=emp.pdf desformat=PDF
```

Applying an XML Customization File Using PL/SQL

To apply an XML report definition to an .RDF file in PL/SQL, use the `SRW.ADD_DEFINITION` and `SRW.APPLY_DEFINITION` PL/SQL supplied packages (built-ins) in the Before Form or After Form trigger.

- **Applying an XML Definition Stored in a File. (Built-In)** To apply XML that is stored in the file system to a report, you can use the `SRW.APPLY_DEFINITION` PL/SQL supplied package in the Before Form or After Form triggers of the report:

```
SRW.APPLY_DEFINITION ('d:\orant\tools\doc60\us\rbbr60\cond.xml');
```

When the report is run, the trigger executes and the specified XML file is applied to the report.

- **Applying an XML Definition Stored in Memory.** To create an XML report definition in memory, add the definition to the document buffer using `SRW.ADD_DEFINITION` before applying it using `SRW.APPLY_DEFINITION`.

Customizing Reports at Runtime with XML

Using Oracle Reports Developer, in Reports6i and higher, you can change the appearance and content of a report at runtime. To do this, merge the report definition files (RDFs), built from XML tags, with existing .RDF files at runtime, and then execute the combination. You can use XML report definitions for other tasks such as:

- Making batch modifications using XML report definitions with the Reports Conversion Tool (RWCON60). RWCON60 is a PL/SQL supplied package (Built-in function).
- Building a complete report definition in XML that can be run by itself, without an existing /RDF file.

The following examples show you how to modify reports using XML at runtime, including:

- Changing visual attributes of an item and changing its format-mask
- Modifying the attributes of non-database-fields
- Creation of multi-language reports from one RDF file
- Creation of a report without any definitions in the RDF file out of an XML file

Before you Start...

Before you can use the following examples 1 through 5, you must create a default report (a tabular report - "select * from emp", select all columns, "Corporate 1" template) that the XML customizations are applied to. You can call it anything you want, but in the following examples, we refer to it as emp_report.

To activate these modifications, it is best to create a file called `modify.xml`, and then copy the XML code of the current example into it. This way, you can always use the following same command line to run the report and modifications:

```
rwr60 report=emp_report userid=scott/tiger customize=modify.xml
```

If you modify the scripts listed below, keep all XML specific restrictions in mind, especially the case-sensitivity of XML.

Customizing Reports with XML, Example 1: Modifying F_EMPNO and Setting Color

The following example modifies the field F_EMPNO and sets its color to red.

```
<report name="emp_report" DTDVersion="1.0">
  <layout>
    <section name="main">
      <field name="F_EMPNO" source="EMPNO" textColor="red"/>
    </section>
  </layout>
</report>
```

Customizing Reports with XML, Example 2: Changing Text Color of F_EMPNO

Example 2 changes the text-color of field F_EMPNO to red and sets the date-format of field F_HIREDATE to german notation.

```
<report name="emp_report" DTDVersion="1.0">
  <layout>
    <section name="main">
      <field name="F_EMPNO" source="EMPNO" textColor="red"/>
      <field name="F_HIREDATE" source="HIREDATE" formatMask="dd.mm.yyyy"/>
    </section>
  </layout>
</report>
```

Customizing Reports with XML, Example 3: Modifying Boilerplate Text Objects

This example shows you how to modify boilerplate text objects. Normal report layout elements are enclosed by the `<layout>...</layout>` tags, while the boilerplate and logic elements are enclosed by the `<customize>...</customize>` tags.

```
<report name="emp_report" DTDVersion="1.0">
  <layout>
    <section name="main">
      <field name="F_EMPNO" source="EMPNO" textColor="red"/>
      <field name="F_HIREDATE" source="HIREDATE" formatMask="dd.mm.yyyy"/>
    </section>
  </layout>
  <customize>
    <object name="B_HIREDATE" type="REP_GRAPHIC_TEXT">
      <properties>
        <property name="textSegment"> Anst.Dat. </property>
      </properties>
    </object>
  </customize>
</report>
```

Customizing Reports with XML, Example 4: Replacing a SELECT * Query

As any other element of a report definition, the query is an element that can be customized by the XML file. In this example, the query used when creating the report (select * from emp) is replaced by one using a WHERE-clause (select * from emp where deptno = 10).

Note that you have to use the same datasource name as originally used in the report definition done by the wizard. Otherwise, there would be another datasource created but not used, as there is no association to any repeating frame.

```
<report name="emp_report" DTDVersion="1.0">
  <data>
    <dataSource name="Q_1">
      <select>
        select * from emp where deptno = 10
      </select>
    </dataSource>
  </data>
  <layout>
    <section name="main">
      <field name="F_EMPNO" source="EMPNO" textColor="red"/>
      <field name="F_HIREDATE" source="HIREDATE" formatMask="dd.mm.yyyy"/>
    </section>
  </layout>
</report>
```

```

    </section>
</layout>
<customize>
  <object name="B_HIREDATE" type="REP_GRAPHIC_TEXT">
    <properties>
      <property name="textSegment"> Anst.Dat. </property>
    </properties>
  </object>
</customize>
</report>

```

Customizing Reports with XML, Example 5: Adding a Trigger to Field S_SAL

Besides changing the visual attributes of a report, you can easily modify its logic by customizing the PL/SQL code as well. In this example, the report we created does not contain format-triggers. By applying this XML file, a trigger is added to field, F_SAL, that hides the field if the value of :SAL is less than 2500.

```

<report name="emp_report" DTDVersion="1.0">
  <layout>
    <section name="main">
      <field name="F_EMPNO" source="EMPNO" textColor="red"/>
      <field name="F_HIREDATE" source="HIREDATE" formatMask="dd.mm.yyyy"/>
      <field name="F_SAL" source="SAL" formatTrigger="SAL_FORMAT"/>
    </section>
  </layout>
  <programUnits>
    <function name="SAL_FORMAT">
      <![CDATA[
        function sal_format return boolean
        is
        begin
          if :SAL > 2500 then
            return (true);
          else
            return (false);
          end if;
        end;
      ]]>
    </function>
  </programUnits>
  <customize>
    <object name="B_HIREDATE" type="REP_GRAPHIC_TEXT">
      <properties>

```

```
        <property name="textSegment"> Anst.Dat. </property>
    </properties>
</object>
</customize>
</report>
```

Performing Batch Report Modifications by Applying XML Report Definitions

Reports' ability to externalize modifications simplifies upgrading and the need to make an application site-specific. You can apply an XML definition and save the resultant definition as a new unique module. This also facilitates updates or upgrades without having to open each file in Reports Builder to make changes.

To update a large number of reports, you can use the CUSTOMIZE command line argument with `RWCON60`, the Reports Conversion Tool, to perform modifications in batch. Batch modifications are useful when making repetitive changes to a large number of reports, for example, when changing a field's format mask. From Oracle Report Builder, you can run `RWCON60` once and make the same change to a large number of reports.

The following example applies two XML report definitions, `translate.xml` and `customize.xml`, to three .RDF files, `inven.rdf`, `inven2.rdf`, and `manu.rdf`.

It saves the revised definitions to new files:

- `inven1_new.rdf`
- `inven2_new.rdf`
- `manu_new.rdf`

```
rwcon60 username/password@mydb
  stype=rdffile
  source="(inven1.rdf, inven2.rdf, manu.rdf)"
  dtype=rdffile
  dest="(inven1_new.rdf, inven2_new.rdf, manu_new.rdf)"
  customize="(e:\apps\trans\translate.xml,
  e:\apps\custom\customize.xml)" batch=yes
```

Creating Mutated RDFs Out of One Master

For example, ERP vendors may want each customer to individualize their own reports. This requires modifying fonts, colors, and so on. By using the Reports XML based customization, vendors can make changes like this to the entire application in one step.

To do so, simply apply the XML customization files using `RWCON60` and create a new RDF or REP file out of it, like you can see in the example above. This way, you can create customized applications for *each* customer without the problem of maintaining different versions of a report. You only have one master-report containing your basic layout and maybe placeholder objects for customer-specific things like logos, company names. And then you apply a customization file for each customer building their own report.

Creating Multi-Version Reports Out of a Single RDF

Creating multi-language reports is always a challenging task. Developing a base report and then translating it for all supported languages can cause maintenance problems. With OracleAS Reports Services' XML customization feature, this is easily done.

You create your report in the base-language and then apply different XML customization files containing language specific settings, such as, label text, data-format, and numeric formats. This way you can create different language versions of your report easily.

Customizing Reports with XML Example 6: Creating Different Language Versions

This example shows how you can easily create different language versions from one report definition. You no longer have to do heavy coding or multiple versions of the same report. Simply create one layout and localize it by applying a "language-XML".

The example modifies `emp_report` to have a German boilerplate and date-formats. You can easily change this to your local language. Simply replace the German terms by ones used in your language and you have localized the report.

```
<report name="emp_report" DTDVersion="1.0">
  <layout>
    <section name="main">
      <field name="F_EMPNO" source="EMPNO" textColor="red"/>
      <field name="F_HIREDATE" source="HIREDATE" formatMask="dd.mm.yyyy"/>
      <field name="F_DATE1_SEC2" source="Current Date" formatMask="dd.mm.yyyy"/>
    </section>
  </layout>
</report>
```

```
</section>
</layout>
<customize>
  <object name="B_EMPNO" type="REP_GRAPHIC_TEXT">
    <properties>
      <property name="textSegment"> Pers.No. </property>
    </properties>
  </object>
  <object name="B_ENAME" type="REP_GRAPHIC_TEXT">
    <properties>
      <property name="textSegment"> Name </property>
    </properties>
  </object>
  <object name="B_JOB" type="REP_GRAPHIC_TEXT">
    <properties>
      <property name="textSegment"> Pos. </property>
    </properties>
  </object>
  <object name="B_MGR" type="REP_GRAPHIC_TEXT">
    <properties>
      <property name="textSegment"> Vorges. </property>
    </properties>
  </object>
  <object name="B_HIREDATE" type="REP_GRAPHIC_TEXT">
    <properties>
      <property name="textSegment"> Anst.Dat. </property>
    </properties>
  </object>
  <object name="B_SAL" type="REP_GRAPHIC_TEXT">
    <properties>
      <property name="textSegment"> Geh. </property>
    </properties>
  </object>
  <object name="B_COMM" type="REP_GRAPHIC_TEXT">
    <properties>
      <property name="textSegment"> Prov. </property>
    </properties>
  </object>
  <object name="B_DEPTNO" type="REP_GRAPHIC_TEXT">
    <properties>
      <property name="textSegment"> Abt. </property>
    </properties>
  </object>
  <object name="B_DATE1_SEC2" type="REP_GRAPHIC_TEXT">
    <properties>
```

```

    <property name="textSegment"> Stand vom </property>
  </properties>
</object>
</customize>
</report>

```

If your application requires dynamic switching of languages, you can also consider applying the customization at runtime.

Creating Report Definitions in XML

Another use of XML report definitions is to make an entire report definition in XML that can be run independently of another report. The advantage of this is that you can build a report without using the Oracle Report Builder. In fact, you could even use your own front end to generate the necessary XML and allow your users to build their own reports dynamically.

With OracleAS Reports Services you can also create a report in Report-Builder, save it in XML format so that you have a starting point. You can then modify the XML or use it as template for an application that creates the XML.

Customizing Reports with XML, Example 7: Report from XML Definitions Only

This example requires an empty RDF file. Just create an empty report, and save it as an empty .rdf. You then apply the following XML that contains the needed modifications to create a report out of XML definitions only.

```
rwrunc60 report=empty userid=scott/tiger customize=modify.xml
```

This example creates a simple report that displays the columns EMPNO, ENAME, SAL, and COMM using the template corp1.tdf. This report looks exactly like one created using the report-wizard.

```

<report name="emp_report" DTDVersion="1.0">
  <data>
    <dataSource name="Q_EMP">
      <select>
        select empno, ename, sal, comm from emp
      </select>
    </dataSource>
  </data>
  <layout>
    <section name="main">
      <groupLeft name="M_emp" template="corp1.tdf">

```

```
<group>
  <field name="F_EMPNO" source="empno"/>
  <field name="F_ENAME" source="ename"/>
  <field name="F_SAL" source="sal"/>
  <field name="F_COMM" source="comm"/>
</group>
</groupLeft>
</section>
</layout>
</report>
```

Running XML Report Definitions

Once you have created your XML report definition, you can use it in the following ways.

- **"Applying an XML Customization"**. Apply XML report definitions to .RDF or other .XML files at runtime by specifying the CUSTOMIZE command line argument or the SRW.APPLY_DEFINITION built-in.
- **"Running an XML Report Definition by Itself"**. Run an XML report definition by itself (without another report) by specifying the REPORT command line argument.

Running an XML Report Definition by Itself

To run an XML report definition by itself, send a request with an XML file specified in the REPORT argument. You can do this in the following ways:

- From the command line, to send a job request to OracleAS Reports Services to run report, emp.xml, by itself, use:

```
rwcli60 userid=username/password@mydb
report=e:\corp\myreports\emp.xml
destype=file desname=emp.pdf desformat=PDF
server=repserver
```

- From OracleAS Reports Services Runtime, to send the equivalent job request, the command would be:

```
rwrun60 userid=username/password@mydb
report=e:\corp\myreports\emp.xml
destype=file desname=emp.pdf desformat=PDF
```


When running an XML report definition in this way, the file extension must be `.XML`. You could apply an XML customization file to this report using the `CUSTOMIZE` argument.

XML Used in JSP for Storing Report Definitions

In Reports 9i, besides using RDF and XML, you can also use Java Server Pages (JSP), as a format for saving a report. Inside the JSP, report elements such as data-model and paper-layout, are stored in XML format.

Depending on what you want to do with the JSP, it may contain only the data-model and the web-source, or it may also contain the paper-layout. In this case you can produce, for example, a PDF document from the same file.

Using XML as a Datasource

OracleAS Reports Services introduces the concept of a pluggable data source (PDS). PDS enables you to create interfaces to your own data sources, hence allowing reports to access this data and use it together with data from other PDSs in a single report.

Through a published interface, OracleAS Reports Services communicates with the PDS and uses it to fetch data from the specified source. Reports PDSs are transparent to the user. The Reports pluggable data sources (PDS) can be used side-by-side in the same data model and linked together.

The PDS is written in JAVA™ and then linked into OracleAS Reports Services using the configuration files.

Pluggable Data Source, XML-PDS

The XML-PDS is one of the PDSs shipped with OracleAS Reports Services. It enables you to access XML-data from a file or live stream from the Internet. The structure of the XML data must follow a DTD (document type definition) or XSD (XML schema definition). The advantage of using the XML schema is:

- With XML Schema Definition, the columns can have different data types
- With a normal XML document using a DTD, only the structure and simple syntax-specific information is stored. All values are of type CHARACTER.

Using XML for OracleAS Reports Services Configuration Files

As the configuration-files for Oracle Application Server – Reports Service became more and more complex, their format has moved to XML for easier modification and readability.

Pluggable Data Source (PDS) Configuration

Needing to store configuration information, the XML-PDS and the JDBC-PDS both use XML-files for storing their preference settings.

Server Configuration

The configuration file for the server has also moved to XML format. When you migrate from an older version, such as, Reports 6i, the server will read the old configuration file and create a file in XML format for you.

Here is an example of a simple configuration file:

```
<?xml version = '1.0' encoding = 'ISO-8859-1'?>
<!DOCTYPE server PUBLIC "-//Oracle Corp.//DTD Reports Server Configuration
9i//EN" "file:/d:/orawin70/report70/server/jasmine.dtd">
<server>
  <cache class="oracle.reports.cache.RWCache">
    <property name="cacheSize" value="50"/>
    <property name="cacheDir" value="d:\orawin70\report70\server\cache"/>
  </cache>
  <!--Please do not change the id for reports engine.-->
  <!--The class specifies below is subclass of _EngineImplBase and implements
EngineInterface.-->
  <engine id="rwEng" class="oracle.reports.engine.EngineImpl" initEngine="1"
maxEngine="1" minEngine="0" engLife="50" maxIdle="30" callbackTimeOut="60000">
    <property name="cacheDir" value="d:\orawin70\report70\server\cache"/>
  </engine>
  <job jobType="report" engineId="rwEng"/>
  <log option="noJob"/>
  <trace traceOpts="trace_all" traceFile="foo.txt" traceMode="trace_replace"/>
  <connection maxConnect="20" idleTimeOut="15">
    <orbClient id="RWClient" publicKeyFile="clientpub.key"/>
    <cluster publicKeyFile="serverpub.key" privateKeyFile="serverpri.key"/>
  </connection>
  <queue maxQueueSize="10000"/>
  <persistFile fileName="d:\orawin70\report70\server\demo-pc.c7.dat"/>
</server>
```

Distribution File

The distribution capabilities in OracleAS Reports Services have been enhanced dramatically. To fit these changes and make the management of the distribution file easier, it also now uses the XML format.

The new distribution files now look like the following:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<destination>
  <file id = "1" name = "Testfile.html" section = "main" format = "htmlcss">
    <include src = ""/>
  </file>
  <mail id = "2" to = "test@myserver.com">
    <attach format = "pdf" srctype = "report">
      <include src = "test.pdf"/>
    </attach>
  </mail>
</destination>
```

How Reports9i XML-PDS Supports XSQL Servlet

XML-PDS allows you to specify an XSQL Servlet file as the XML data source.

You can specify the URL (local or remote) of the XSQL file, as the data source URL for the XML- PDS. XML-PDS then sends the URL request to the Web Server. To process the XSQL file, the webserver noted in the URL, must be configured with XSQL Servlet. By identifying the specific extension of the file mentioned in the URL, such as .xsql in this case, (which can be configured in the Webserver), it invokes XSQL Servlet.

XSQL servlet has the information on the database connection in an XSQL Pages configuration file. It processes the given XSQL Page, sends the SQL query to the database through a JDBC interface, receives the resultset , and puts the resultset in XML format. XSQL Servlet sends this to XML-PDS. Hereafter XML-PDS treats this XML as any other XML data source and processes it inside the reports.

An XML Schema or DTD must be used as Data Definition when XSQL is used as Data Source.

Reports Case Studies

How to Become a Supplier of Live XML Streams

In the ever-growing B2B environment, time has become a vital factor. Many businesses rely on just-in-time delivery to avoid large inventories and the need for large amounts of fixed capital. Hence, it is vital for merchants to know, what their suppliers have in stock at any moment.

To offer such a service, suppliers could provide their inventory-information on their web-page. However, in most cases this information is needed in a format that can be used for further processing; a format such as XML. Even more useful would be a tool that can produce a report in HTML, PDF, and XML, out of the same report. So, merchants can either:

- Go to the web-page of their suppliers and look up the information.
- Use a URL and get the information in XML format to import it into their own system or even the system accessing the data at the moment of request.

In this case, Oracle Application Server – Reports service offers an ideal solution. You can design a good looking report for creating HTML or PDF output, and at the same time use this to produce an XML stream.

Figure 14–4 shows an example of a report produced by OracleAS Reports Services in HTML format.

Figure 14–4 Example Report of Inventory Data Producing HTML and PDF Output

Id	Title	In Stock	Back Order
1000001	Hat Parisienne brown	15	10
1000002	Baseball Cap Blue	34	

Note: The link in the upper left corner of the report-output. It calls the report itself but with destination-type XML and produces an XML-format output from the same report.

Figure 14-5 shows an example report representing the same inventory data in Figure 14-4 but as an XML stream.

Figure 14-5 Example Report For Inventory Data as an XML Stream

```

<?xml version="1.0" ?>
<!-- Generated by Oracle Reports version 6.0.8.13.1 -->
- <INVENTORY>
- <INVENTORY_ITEM>
  <ID>1000001</ID>
  <TITLE>Hat Parisienne</TITLE>
  <SUBTITLE>brown</SUBTITLE>
  <IN_STOCK>15</IN_STOCK>
  <BACK_ORDER>10</BACK_ORDER>
  <BACKORDER_AVAILABLE>15-APR-01</BACKORDER_AVAILABLE>
  <ITEM_CANCELED_SINCE />
  <PRICE_PER_UNIT />
  <UNITS_PER_PACKAGING />
  <PACKAGE_DESCRIPTION />
</INVENTORY_ITEM>

```

How to Take Advantage of Supplied XML-Data

As merchants, you can use the inventory data provided by your suppliers and combine it with data your own inventory to create a virtual stock list.

OracleAS Reports Services now also enables you to use:

- SQL-PDS (SQL-Pluggable Data Source) module to access your inventory data
- XML-PDS (XML-Pluggable Data Source) module to access your inventory data of your suppliers

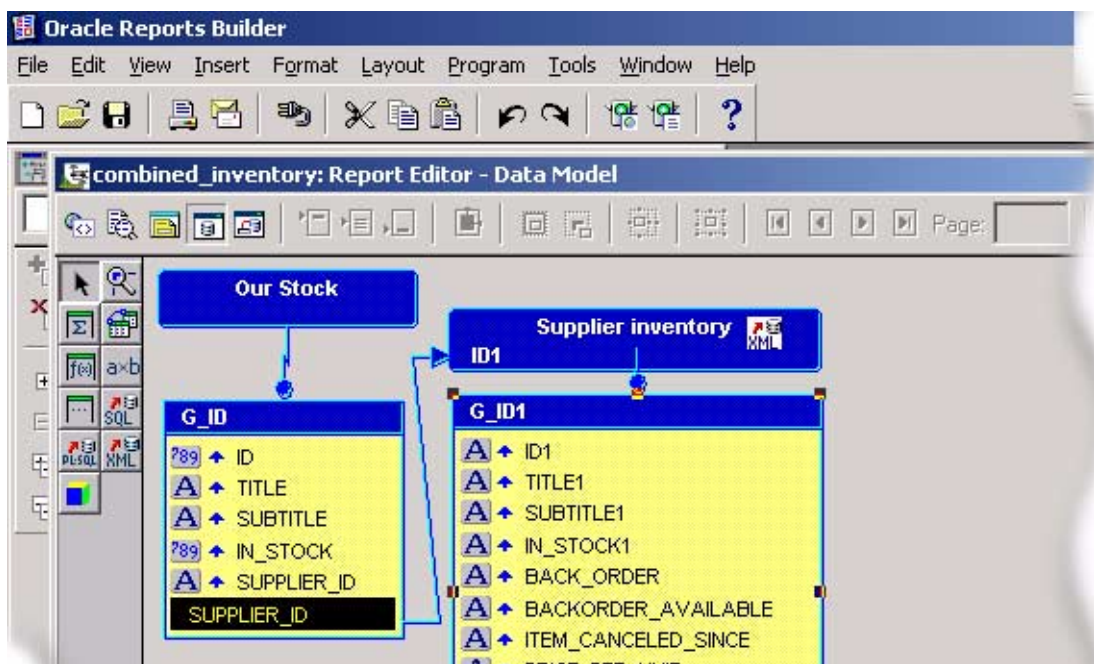
To do so, you, ideally need the URL to your suppliers XML-stream and a DTD or XML schema definition that describes the data provided by the XML stream. For example, the DTD would look something like the following:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!ELEMENT INVENTORY (INVENTORY_ITEM+)>
<!ELEMENT INVENTORY_ITEM
  ( ID ,
    TITLE ,
    SUBTITLE ,
    IN_STOCK ,
    BACK_ORDER ,
    BACKORDER_AVAILABLE ,
    ITEM_CANCELED_SINCE ,
    PRICE_PER_UNIT ,
    UNITS_PER_PACKAGING ,
    PACKAGE_DESCRIPTION
  )>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT SUBTITLE (#PCDATA)>
<!ELEMENT IN_STOCK (#PCDATA)>
<!ELEMENT BACK_ORDER (#PCDATA)>
<!ELEMENT BACKORDER_AVAILABLE (#PCDATA)>
<!ELEMENT ITEM_CANCELED_SINCE (#PCDATA)>
<!ELEMENT PRICE_PER_UNIT (#PCDATA)>
<!ELEMENT UNITS_PER_PACKAGING (#PCDATA)>
<!ELEMENT PACKAGE_DESCRIPTION (#PCDATA)>
```

As the PDS is transparent to you, different PDSs (in this case, SQL and XML PDSs), work seamlessly together in the data-model and can be handled as if they are SQL-ones (that is, joined together).

Figure 14-6 shows a data-model in Oracle Report Builder, with both the SQL and XML data sources linked together.

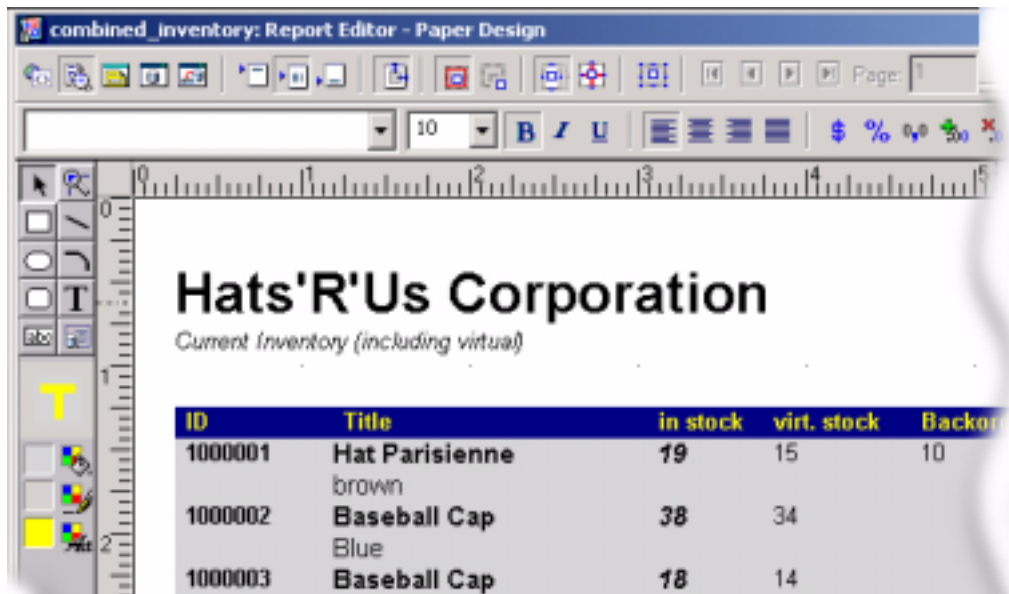
Figure 14–6 Report Builder: A Data Model Showing the SQL and XML Data Sources Linked Together



You can also create a layout using data from both the data sources together. In this case, the columns, 'virtual stock' and 'backorder' come from the XML-stream and represent data for this particular item in your one supplier's inventory. The data is fetched everytime the report is run using a URL to a report on the supplier's site. This produces an XML stream of your supplier's inventory data.

Figure 14–7 shows the finished report of inventory data that combines SQL data, for example, retrieved from the merchants own inventory, and data from a remote XML stream (virtual data), for example, of inventory data retrieved from a supplier.

Figure 14–7 Example Report of Inventory Data Combining SQL Data and Data From a Remote XML Stream



Frequently Asked Questions: Reports and XML

Can We Output XML From Our Year End Reports Through a Database Interface?

We are working on AU/SG Year End Reporting which involve archiving and production of magtape files. Do you have information on EOY reporting? Have you created any sites/documentation that explain how XML is actually being used for EOY reporting? And what it's being used for? We are using DBIs in fast formulas to obtain a lot of the YE information. Can we use/access DBIs in XML?

Answer

You can output XML from Reports 6i by just changing a single parameter - DESFORMAT. Instead of it being set to HTML or PDF,... just change it to XML. This will work with your existing report, so you should not have to do anything extra.

What you describe after generating the XML output (that is, applying a stylesheet) is trivial in reports - there is an undocumented PL/SQL built-in `SRW.SET_XML_`

PROLOG to allow you to set the XML prolog line. Refer to bug 1265291 for information on how to do this. It should take about 5mins to test.

With Reports you could have nice looking printable output (PDF, postscript, and so on) or on the web in HTML or HTMLCSS - but at the same time, get XML out just by changing a single parameter, and then use in a B2B environment such as the one described.

Regarding XML report generation, you can do this in different ways:

- Using OracleAS Reports Services and an existing report structure. Generate XML as the output for the report (an Oracle Reports 6i feature). Then to apply a stylesheet for creating the totals and formatting the report following the schema given.
- Using Java and the Oracle XML SQL Utility (XSU). In this case a Java Stored Procedure is created for retrieving all the information needed in the format required. Information about the Oracle Developer Kit is at <http://otn.oracle.com/tech/xml/>
- Using the XSQL Servlet. You may find very useful information at http://larva.us.oracle.com/docs/tech/xml/oracle_xsu/doc_library/relnotes.html
- Using the current Magtape process. The magtape process already generates all the data necessary. The problem is that no XML code is produced. The solution in this case would involve the creation of a new file (in XML format) using the standard fast-formula strategy for the magtape process.

About your question on DBIs, you can access any table/view in the database using the XML developer kit or the XSQL Servlet, so you can generate XML code using the information from the database items (FF_DATABASE_ITEMS table)

Changing the Report Template

I have a report with a template. The same report is needed with another template. I copied the report and try to change the template. Is there another way of changing the template than the wizard, because the wizard changes the layout of the report and I do not want that.

Answer

You must apply the template first, because when you use the Report Wizard to apply a new template, it will then create a new layout overriding the existing layout. There is no other way.

However, you can use XML for modifying the Report at runtime without changing the layout. See this chapter under section, "[Customizing Reports at Runtime with XML](#)" on page 14-8, for some simple examples for modifying a report using XML at runtime.

REP-6106:Error in the XML report definition at line 1

I tried to apply XML to my RDF file to change the boilerplate text from English to Chinese. My XML is shown as below:

```
<report name="am01.rdf" DTDVersion="1.0">
<layout>
<section name="main">
<field name="f_title" source="title" textColor="red" fontSize="16"
fontStyle="bold"/>
</section>
</layout>
<customize>
<object name="B_4" type="REP_GRAPHIC_TEXT">
<properties>
<property name="textSegment"> çÓ*Ãû</property>
</properties>
</object>
</customize>
</report>
```

I saved this XML file as unicode format, and when I tried applying XML to my RDF through RWRUN60 from command line, report builder gives me the following message:

```
REP-6106:Error in the XML report definition at line 1 in 'c:\am01.xml'
Start of root element expected instead of TEXT 'null'
```

I am using Oracle Report Server/Developer 6i running with Apache web server.

Answer

See the following example for ideas on answering your question:

["Customizing Reports with XML, Example 4: Replacing a SELECT * Query"](#)

Using the PDK for Visualizing XML Data in Oracle Portal

This chapter describes the following sections:

- [Introducing Oracle Portal](#)
- [Common Portlet Applications](#)
- [Oracle Portal Development Kit \(PDK\)](#)
- [PDK URL Services](#)
- [PDK URL Services Overview](#)
- [URL Services Architecture](#)
- [Provider.xml](#)
- [Configuring provider.xml](#)
- [Integrating Technologies into OracleAS Portal](#)

Introducing Oracle Portal

OracleAS Portal is a component of Oracle. It offers the security, reliability, and scalability of Oracle Application Server (OracleAS) and Oracle database. It is comprised of easy-to-use portal software and a suite of products for application development, data warehousing, business intelligence, application integration, and mobile computing.

What are Portlets?

A portlet is a contained region on an Oracle Portal web page. Portlets can be considered as "web components" that display excerpts of other web sites and generate summaries of key information. Portlets can be placed on the same page with other portlets so that users have easy access to frequently used sites and information.

Portlets are rendered by web browsers just like any other part of a web page. Typically, portlets use standard HTML to display information to users, but their interfaces can be extended using other browser-capable technologies such as Cascading Style Sheets (CSS), eXtensible Style Language (XSL), JavaScript, and even Java applets.

Portlets can be used to access nearly any type of web-accessible information --- from files stored on the corporate intranet and reports on data managed by corporate applications to news and stock quotes from the Internet. Because of their dynamic nature, portlets are often used to highlight important information, alert users to new developments, and summarize key data.

There are three types of portlets:

- **Built-in portlets.** Oracle Portal provides a set of built-in portlets with ready-to-use functionality for web application development, web publishing and external site integration.
- **Database Portlets.** Implemented as stored procedures and executed in the database. Can be written in PL/SQL or Java Stored Procedures wrapped in PL/SQL. DATABASE PORTLETS. Use Database Portlets whenever your portlets require significant database interaction or when the development team has Oracle experience. To create a Database Portlet:
 - Create a Database Provider by creating a package that exposes methods required by the API to display portlets accordingly.
 - Code the portlet producing any technology that can be rendered within an HTML table cell, including HTML, JavaScript, applets and certain plug-ins.

- Register the provider with the Portal before it can be accessed and used. This step refreshes the portlet repository.
- Refresh the portlet repository which stores information for the providers and portlets that the provider owns. You will need to do this manually if you change portlet information after registering your provider.
- **Web Portlets.** Implemented using any web-capable language on a web server external to the Portal. The PDK includes Java classes to make building web portlets easier.

Common Portlet Applications

Here are some common applications for portlets:

- **Centralizing access to intranet sites.** Oracle Portal makes it easy to gather links to all many sites in one place, then organize these access points so that users have a simple way to find what they are looking for. Sets of these links can be published as portlets so that users can readily access frequently used sites from their own personalized page.
- **Publishing information and documents.** Content on the web can be arranged into folders, then placed as portlets onto any portal page, allowing you to mix and match the specific content you wish to see.
- **Integrating dynamic data services.** Portlets are useful for rendering content provided from external data sources and displaying that information on portal pages. For example, portlets allow real-time news stories to be displayed within the portal from XML data sources.
- **Providing an interface to web applications.** Portlets can be used to automatically login to well used application (or its data store) and retrieve a summary of that information. The portlet can then display the information on a portal page.
- **Integrating with other corporate systems.** Organizations have many different, sometimes incompatible, systems. Portlets allow the interfaces for these systems to be presented in a consistent manner within the Oracle Portal environment.

Oracle Portal Development Kit (PDK)

Oracle Portal Development Kit (PDK) includes services and tools for extending the OracleAS Portal framework by developing portlets. These services include:

PDK Integration Services (PDKIS)

PDK Integration Services (PDKIS) is one of the services offered by OracleAS Portal. PDKIS allow you to pull content into portlets directly from URLs, including URLs requiring authentication before viewing. PDKIS can do either of the following tasks:

- Parse HTML content directly and place it into a portlet
- Transform the HTML content into XHTML for further processing using an XSL stylesheet

PDKIS can pull content into your portal, and then modify the default XSL stylesheet to select what to display and how to display it.

PDK URL Services

You can extend the JPDK (PDK for Java) to create URL-based portlets in any language. Create portlets using any existing application without altering any code. These services can be installed on any machine using JPDK 1.4 or later.

What's Needed to Run URL Services

You need the following to run URL Services:

- OracleAS Portal 3.0.8.9.8 or later. Most features of the PDK URL Services work with older versions, but is only certified against this version of OracleAS Portal. Certain features such as authenticated portlets will not work on older versions of OracleAS Portal.
- JPDK 1.4 or later.

PDK URL Services Overview

Oracle Portal Development Kit (PDK) currently provides services for Java and PL/SQL. These services allow developers to integrate Java classes and servlets, Java Server Pages, and PL/SQL as portlets within OracleAS Portal using Portal APIs. The PDK and JPDK (PDK Services for Java) provide samples, utilities, and articles to easily develop portlets in PL/SQL and Java, but do not provide a simple solution for developers who have applications written in any other language like C, C++, Perl, ASP,... To simplify developing portlets in any language, Oracle Portal Development Kit provides PDK URL Services.

PDK URL Services allow developers to take any application written in any language and easily create integrated portlets. The URL Services takes the URL of

an application, parses the content, and uses the JPDK framework to create a portlet. This process allows each 'show mode' of your portlet to be rendered from different applications and/or languages. For example, a portlet can have 'show mode' rendered using PERL, 'edit mode' rendered using ASP, 'help mode' rendered using HTML, 'details mode' rendered using JSP, and so on.

- The current JPDK Framework allows you to define and list portlets through the `provider.xml` file and limits the amount of programming required.
- PDK URL Services extends the JPDK Framework and takes advantage of its ease and simplicity. Therefore, creating URL portlets follows the same steps and configuration needed to create Web portlets with a few exceptions.

Creating a URL Portlet

To create a URL portlet, follow these main steps:

1. Create your application in any language.
2. Configure your application to be accessible through a URL.
3. Define your application through the `provider.xml` by providing a URL.
4. Register your provider as a "Web" provider through OracleAS Portal.
5. Add the portlet to a page.

Web Provider

A Web Provider is one that is written as a Web application. It is installed and hosted in a Web server and is remote from Oracle Portal. A Web Provider also owns and manages a set of portlets. PDK Java Services offers a Provider Runtime called `DefaultProvider` that implements the functions of a provider. `DefaultProvider` owns and manages a set of portlets. It uses a initialization file to manage the set of portlets, called `provider.xml`.

The file, `provider.xml`, is a static file that stores information about a provider and its portlets. Understanding the configuration of `provider.xml` allows you to create your own file to list and describe your Web portlets.

URL Services Architecture

URL Services uses existing JPDK classes and extends the framework where required for rendering content from a URL. PDK URL Services allows you to define and list your URL portlets within the `provider.xml` file.

URL Services also eliminates the need for additional programming by including a default runtime that handles portlet creation, integration, and communication with OracleAS Portal. The three main components of PDK URL Services are:

- URL Services Interface
- URL Services Runtime
- `provider.xml`.

URL Services Interface

URL Services Interface is additions to the Web Provider Interface included in JPDK. The interface specific to URL rendering is `oracle.portal.provider.v1.ContentFilter`, specifications for filtering URL content.

URL Services Runtime

URL Services Runtime extends classes from the current Web Provider (JPDK) Runtime to adapt to the new URL rendering capabilities. URL Services Runtime is comprised of the following set of runtime classes:

- `oracle.portal.provider.v1.http.DefaultURLProvider` extends the Web Provider Runtime class `DefaultProvider`. This class represents the Provider for all portlets rendered using the URL Services.
- `oracle.portal.provider.v1.http.PortletNodeHandler` extends the Web Provider Runtime class `DefaultPortlet`. This class parses all of the XML content within the `provider.xml` referring to URL rendering.
- `oracle.portal.provider.v1.http.URLSecurityManager` implements the Web Provider Interface defined by `oracle.portal.provider.v1.PortletSecurityManager`. This class manages portlet access and security.
- `oracle.portal.provider.v1.http.URLPageRenderer` implements the Web Provider Interfaces defined by `oracle.portal.v1.PortletRenderer`. The `URLPageRenderer` allows you to use the URL to render the content of a request.
- `oracle.portal.provider.v1.http.XMLFilter` implements the Web Provider Interface defined by `oracle.portal.provider.v1.ContentFilter`. This filter converts the HTML contents received from a URL into XHTML.
- `oracle.portal.provider.v1.http.HtmlFilter` implements the Web Provider Interface defined by `oracle.portal.provider.v1.ContentFilter`. This filter converts the HTML contents into OracleAS compliant HTML.

DefaultXhtml.xsl is a default stylesheet that converts XHTML to OracleAS Portal compliant XHTML.

Provider.xml

provider.xml stores information by hierarchy and defines and lists available portlets. provider.xml is associated with only one provider. The default provider in the XML file is oracle.portal.provider.v1.http.DefaultProvider. To take advantage of URL Services, you need to specify oracle.portal.provider.v1.http.DefaultURLProvider. DefaultURLProvider parses the additional/updated tags within provider.xml file. The DefaultURLProvider can handle standard provider.xml tags, but DefaultProvider cannot handle tags within the provider.xml that contain URL Services information.

provider.xml Tags

Table 15-1 lists provider.xml tags that have been added or modified in the provider.xml:

Table 15-1 provider.xml Tags

provider.xml Tags	Description
provider tag	The following are tags that have been added or modified within the provider tag.
authentication	A required tag that has been added to the provider tag. It holds information about the type of authentication used.
proxyInfo	A required tag and holds proxy server information.
authorizatio	A required tag and holds information about the type of authorization.
redirectUrl	An optional tag and contains the parameter name used by the External Application for redirection after a successful authentication.
portlet tag	The following are tags that have been added or modified within the portlet tag.
registrationPortlet	An optional tag and specifies whether this portlet is a registration portlet for the provider.

Table 15–1 *provider.xml* Tags

provider.xml Tags	Description
portletRenderer	Modified to accept URL pages and filters for each show mode.

Using provider.xml

`Provider.xml` is a declarative file containing descriptive informatio. It is used to list and display portlets. `DefaultProvider` parses `provider.xml` to gather information from the file. It creates a portlet instance (Java object) for each portlet listed in `provider.xml`. `DefaultProvider` also retrieves the render modes, personalization, and security information from `provider.xml`. It attaches this information to each of the portlet instances it creates and pushes the information to the other class files.

`DefaultProvider` pushes the information about render modes to the `PortletRender`. It pushes the information about personalization to the `PortletPersonalizationManager`. Finally, it pushes security information to the `PortletSecurityManager`. This allows each class to retrieve information from `provider.xml` without knowing its name or location.

Once `DefaultProvider` parses `provider.xml`, it stores the information until the instance is shut down. When updating, adding, or removing information from `provider.xml`, you must stop and restart the Oracle HTTP Server and also refresh the Portal repository.

XML parser preserves whitespace surrounding XML elements that contain text, so you must take care with the use of whitespace and line feeds when editing `provider.xml`. Consider the example:

```
<showEdit> true </showEdit>
<showEditDefault> true </showEditDefault>
<hasHelp> true </hasHelp>
```

Here, the boolean string values in these tags would not be recognized because of the surrounding whitespace and therefore would be evaluated as false. The tags should be specified as follows:

```
<showEdit>true</showEdit>
<showEditDefault>true</showEditDefault>
<hasHelp>true</hasHelp>
```

Configuring provider.xml

`provider.xml` stores information that is used by the Web Provider. It stores information by hierarchy and its hierarchical nature starts with "Provider". The file is organized to simplify the process required by the DefaultProvider to parse the file. `provider.xml` file is also organized for readability.

This section describes the information in `provider.xml`. When creating your own `provider.xml`, you must follow the hierarchy and syntax required for the DefaultProvider to properly parse the file.

Provider Tag

The provider tag is the first tag within `provider.xml`. It specifies the class that implements `oracle.portal.provider.v1.Provider`. This specification points the Provider Adapter to a corresponding Provider. The provider tag has two attributes:

- `class` is an optional attribute and names the Java class that implements `oracle.portal.provider.v1.Provider`. If no class is specified, it defaults to `oracle.portal.provider.v1.http.DefaultProvider`.

`session` is an optional attribute and is used to disable the `initSession` method within the DefaultProvider. If this session is disabled, the DefaultProvider does not create a servlet session. The default for this attribute is `true`.

Below is a sample of the provider tag from `provider.xml`. In the sample, the provider tag declares DefaultProvider as the class that implements `oracle.portal.provider.v1.Provider` and that DefaultProvider creates a servlet session.

```
<provider class="oracle.portal.provider.v1.http.DefaultProvider" session="true">
```

The provider tag manages a single tag: `useOldStyleHeaders` is an optional. Set this to `true` if you are using Oracle Portal 3.0.6.6.5 with JPKD 1.3 and later, your provider's portlets support `Customize/Help/About` links and you want to retain the 'old 3.0.6 style' headers and footers.

For example, include the following line under the provider

```
tag: <useOldStyleHeaders>true</useOldStyleHeaders>
```

Portlet Tag

Web Provider has a tag called `portlet`. There is one portlet tag for each portlet that the provider manages. The portlet tag declares the class that implements

oracle.portal.provider.v1.Portlet. The tag lists and describes a set of portlets that this provider manages. The portlet tag has three attributes:

- class is an optional attribute and names the Java class that implements oracle.portal.provider.v1.Portlet. If no class is specified, it defaults to
- oracle.portal.provider.v1.http.DefaultPortlet.
- resource is an optional attribute and specifies the class file that represents the resource bundle. The resource bundle is a compiled Java source that localizes the portlet string of the meta-data information. The resource bundle stores information about your portlet in a local environment. You can store information like portlet name, portlet title, and portlet description. If you specify information in the resource bundle, you would not create a tag in the provider.xml for those values. Default for this attribute is null.

version is an optional attribute and specifies the PDK version that the portlet implements. It is the version of the Portlet interface the portlet relies on. Currently, the value must be 1 and the attribute defaults to 1 if no version is specified.

Below is a sample of the portlet tag with three attributes from `provider.xml`. In the sample, the portlet tag declares `DefaultPortlet` as the class that implements `Portlet`, declares a resource bundle called `HelloWorldBundle` and a version of 1.

```
<portlet class="oracle.portal.provider.v1.http.DefaultPortlet"  
resource="oracle.portal.sample.devguide.helloworld.resource>HelloWorldBundle"  
version="1" >
```

Portlet tag has twenty tags that it manages. Each tag describes an attribute of the portlet. [Table 15-2](#) lists these tags.

Table 15–2 Configuring provider.xml: Portlet Tags

Portlet Tags	Description
id	A required tag and holds the portlet ID number. This number must be unique within this portlet provider. The number is specified by the developer and does not need to be sequential. There is no limit to the length of the number as it is received in by the DefaultProvider as a LONG. There is no default value for this tag.
name	A required tag and holds the portlet name. This portlet name must be unique and contain no spaces or special characters. There is no default value for this tag.
title	A recommended tag. The title is the display name for the portlet and is what is seen by users accessing your portlet. The title may have spaces and special characters. There is no default value for this tag.
description	A recommended tag. The description is displayed to users adding portlets to a page. There is no default value for this tag.
imageURL	Optional and holds the URL that the portlet image references. There is no default value for this tag.
thumbnailURL	Optional and holds the URL that the thumbnail image references. There is no default for this tag.
timeout	Optional and holds the timeout in seconds that the Portal waits for the portlet before it times out. If no timeout is specified, it takes the timeout of the Provider.
timeoutMsg	Optional and holds the timeout message that displays if the portlet times out. If no timeout message is specified, it takes the timeout message of the Provider.
showEdit	Optional flag for whether the portlet will have an "Customize" link. This is a boolean value and the default for this tag is false.
showEditPublic	Optional flag for whether the public users may edit the portlet. By default a public page where users are not logged-in does not display the "Customize" link. If you want public users to be able to customize the portlet, specify true here. This is a boolean value and the default for this tag is false.
showEditDefault	Optional flag for whether the customization page will have an EditDefaults link. This is a boolean value and the default for this tag is false.
showPreview	Optional flag for whether the portlet will have a preview option when adding portlets to a page. This is a boolean value and the default for this tag is false.
showDetails	Optional flag for whether the portlet can be displayed in a full browser page. This is a boolean value and the default for this tag is false.

Table 15–2 Configuring provider.xml: Portlet Tags

Portlet Tags	Description
hasHelp	Optional flag for whether the portlet will have a "Help" link. This is a boolean value and the default for this tag is false.
hasAbout	Optional flag for whether the portlet will have an "About" link. This is a boolean value and the default for this tag is false.
defaultLocale	Optional tag and holds the language that the portlet uses by default. You specify the Locale as a two digit language and a two digit country. The java.util.Locale class contains a list of locales. For example: en.US
acceptContentTypes	Optional tag and holds mime types that the portlet recognizes. This tag is an array and contains one tag called item.
item	A required tag under acceptContentTypes, it specifies a mime type. There is one item tag per mime type recognized by the portlet. The sample below lists to mime types recognized by the portlet, HTML and XML. <pre><item>text/html</item> <item>text/xml</item></pre>
portletRenderer	A required tag and specifies the class that will render portlet pages. The portletRenderer tag is an array. It has one attribute and eleven tags. class is an optional attribute and names the Java class that implements oracle.portal.provider.v1.PortletRenderer. If no class is specified, it defaults to oracle.portal.provider.v1.http.
PageRenderer: appPath	A required tag and holds the virtual path to the root of pages that render the portlet.
appRoot	A required tag and holds the physical path to the root of pages that render the portlet.
showPage	A required tag and specifies the page that renders the portlet.
aboutPage	An optional tag and specifies the page used to supply information about the portlet. helpPage is an optional tag and specifies the help page of the portlet.
editPage	Optional tag and specifies the page used to display portlet customization.
editDefaultsPage	An optional tag and specifies the page used by an administrator to customize the default settings for the portlet.
previewPage	An optional tag and specifies the preview page of the portlet.
showDetailsPage	An optional tag and specifies the page used to display the portlet in a full browser window.
pageParameterName	An optional tag and holds the parameter name to render additional pages. This tag allows the PortletRenderer to support screen chaining.

Table 15–2 Configuring provider.xml: Portlet Tags

Portlet Tags	Description
renderContainer	<p>An optional tag and is a flag that determines whether to render a title bar and border for this portlet. The default value is true. The sample below declares PageRenderer as the implementation class and displays a render page and a few additional display modes. This portlet also renders a container.</p> <pre><portletRenderer class="oracle.portal.provider.v1.http.PageRenderer" > <appPath>/lottery</appPath> <appRoot>E:\jpd\hdocs\lottery</appRoot> <showPage>lotto.jsp</showPage> <editPage>custom.jsp</editPage> <aboutPage>about.html</aboutPage> <helpPage>help.html</helpPage> <renderContainer>true</renderContainer> </portletRenderer></pre>
portletPersonalizationManager	<p>An optional tag and specifies the class that handles user customization. The portletPersonalizationManager is an array. It has one attribute and two tags.</p>
class	<p>An optional attribute and names the Java class that implements oracle.portal.provider.v1.PortletPersonalizationManager. If no class is specified, it defaults to oracle.portal.provider.v1.http.DefaultPortletPersonalizationManager.</p>
dataClass	<p>An optional tag and references the Java class that implements CustomizationObject. If no class is specified, it defaults to oracle.portal.provider.v1.http.BaseCustomization.</p>
multiLangStringClass	<p>An optional tag and references the Java class that implements the language used to store the customization string. If no class is specified, it defaults to the language of the Java Virtual Machine. The sample below declares DefaultPortletPersonalizationManager as the implementation class, the customization class as BaseCustomization, and language class as HashMLString.</p> <pre><portletPersonalizationManager class="oracle.portal.provider.v1.http.DefaultPortletPersonalizationManager" > <dataClass> oracle.portal.provider.v1.http.BaseCustomization </dataClass> <multiLangStringClass> oracle.portal.provider.v1.HashMLString </multiLangStringClass> </portletPersonalizationManager></pre>
portletSecurityManager	<p>An optional tag that specifies the java class that implements PortletSecurityManager. There is no default value for this tag.</p>

See Also:

- <http://otn.oracle.com/products>
- WebDB 2.1: Getting Started - Installation and Tutorial (A70070-01)
- WebDB 2.1: Creating and Managing Components - Task Help (A74969-01)
- WebDB 2.1: Creating and Managing Components - Field-Level Help (A70072-01) Contains field level help for the component building features of WebDB.
- WebDB 2.1: Creating and Managing Sites - Task Help (A70073-01) Contains task help for the site building features of WebDB.
- WebDB 2.1: Creating and Managing Sites - Field-Level Help (A70074-01)

Integrating Technologies into OracleAS Portal

You can seamlessly integrate OracleAS Portal with technologies not natively included with OracleAS Portal, such as

- Developing portlets with Dynamic Services. See also [Chapter 18, "Using OracleAS Dynamic Services and XML"](#) and <http://otn.oracle.com/products>
- Developing Portlets with HyperText Templates (HTT). HyperText Templates (HTT) is an Oracle developed utility for building dynamic Web Pages using templates. HTT does a 100% separation of presentation, logic, and data. HTT is used by Oracle internal consultants to build Internet and Intranet solutions. HTT is simple to understand and use. The technology facilitates the rapid development of dynamic Web pages and allows for code and template reuse, as well as assembly from multiple templates.

How Oracle Exchange Uses XML

This chapter contains the following sections:

- [Oracle Exchange and XML](#)
- [Stored Transactions](#)
- [Pass Through Transactions](#)
- [XML Delivery Formats](#)
- [E-Business Solution Architecture](#)
- [ATP \(Availability to Promise\) for Oracle Exchange](#)
- [XML Messaging Services](#)

Oracle Exchange and XML

Many Oracle Exchange transactions can be conducted in XML format, if you choose to do so. These transactions include the following:

- Purchase Orders (POs) inbound. POs outbound are not stored on Oracle Exchange.
- Service Orders (SOs) in and outbound
- PO acknowledgments
- SO acknowledgments
- Advance Shipment Notices (ASNs)
- Invoices

You can send and receive documents in XML format through Oracle Exchange using a communication method, such as HTTP with Web methods, or SMTP (e-mail). Oracle Exchange currently supports two transaction models:

- **Stored Transactions.** In this transaction model, transactions are mapped and stored on Oracle Exchange.
- **Pass Through Transactions.** In the second transaction model, Oracle Exchange acts as a mapping and routing hub for documents between suppliers and buyers.

Outbound or Inbound Transactions

Transactions in Oracle Exchange are also labeled as “outbound” or “inbound”, relative to Oracle Exchange.

- **Outbound.** Any document sent from Oracle Exchange to a supplier or buyer is called an “outbound” transaction.
- **Inbound.** Any document generated by a supplier’s or buyer’s system and sent to Oracle Exchange is called an “inbound” transaction.

Stored Transactions

All documents created in Oracle Exchange and inbound purchase orders are stored on Oracle Exchange. These XML documents are mapped and stored in the Oracle Exchange data model:

- **Outbound purchase orders.** When a buyer makes a catalog purchase on Oracle Exchange, Oracle Exchange will send the purchase order to the buyer

via the buyer's selected communication method. Outbound purchase orders are supported in Version 1.0 and later.

- ***Outbound purchase orders from auctions.*** When awarding a Buyer's Auction, the buyer can generate one or more purchase orders for auction items (one purchase order for each supplier to whom auction business was awarded). Oracle Exchange will send the purchase order(s) to the buyer via the buyer's selected communication method. Outbound purchase orders from auctions are supported in Version 5.2 and later.
- ***Inbound purchase orders.*** Purchase orders generated by the buyer's system and sent to Oracle Exchange are called inbound purchase orders. Oracle Exchange will forward the inbound purchase order from the buyer to the supplier via the supplier's selected communication method. Inbound purchase orders are supported in Version 5.2 and later.
- ***Outbound sales orders.*** When a buyer makes a purchase from a supplier's catalog, Oracle Exchange will send the sales order to the supplier via the supplier's selected communication method. Outbound sales orders are supported in Version 1.0 and later.
- ***Outbound purchase order acknowledgments.*** After a supplier has acknowledged a purchase order, Oracle Exchange will update the status of the purchase order and forward the purchase order acknowledgment from the supplier to the buyer via the buyer's selected communication method. Outbound purchase order acknowledgments are supported in Version 1.0 and later.
- ***Inbound purchase order acknowledgments.*** Inbound purchase order acknowledgments are generated by the supplier's system and sent to Oracle Exchange. Oracle Exchange will forward the inbound purchase order acknowledgment from the supplier to the buyer via the buyer's selected communication method. Inbound purchase order acknowledgments are supported in Version 5.2 and later.

Pass Through Transactions

Oracle Exchange acts as a document routing hub for the following transactions. All of the incoming XML documents are mapped and forwarded.

- ***Inbound advance shipment Notices (ASNs).*** The supplier generates an advance shipment notice to inform the buyer about the shipment. ASNs are supported in Oracle Exchange Release 5.2 and later.

- **Outbound advance shipment notices.** Oracle Exchange forwards the supplier's inbound advance shipment notice to the buyer. Outbound ASNs are supported in Version 5.2 and later.
- **Inbound invoices.** The supplier generates an invoice against a purchase order and sends it to Oracle Exchange. Inbound invoices are supported in Oracle Exchange Release 5.2 and later.
- **Outbound invoices.** Oracle Exchange forwards the supplier's inbound invoice to the buyer via the buyer's selected communication method. Outbound invoices are supported in Oracle Exchange Release 5.2 and later.

Buyers and suppliers must set up their system infrastructure to send and receive XML documents through Oracle Exchange.

XML Delivery Formats

Oracle Exchange uses the Open Applications Group (OAG) Extensible Markup Language (XML) to transfer documents. XML is a universal format for structured documents and data (such as spreadsheets, address books, and financial transactions) on the Web. OAG is an independent standards body focused on best practices and process-based XML content for e-Business and application integration.

OAG XML is one particular "flavor" of the XML format. The OAG XML standard is an open standard for defining and transmitting business transactions using XML. Oracle Exchange uses the OAG XML format in an effort to standardize XML transactions by subscribing to the industry consensus-based XML framework for business software application interoperability.

See Also: <http://www.openapplications.org>, for more information about OAG.

E-Business Solution Architecture

Oracle Exchange enables you to use one of the following implementations:

- OMB (Oracle Message Broker) plus adapters
- webMethods

A webMethods implementation is described in "[ATP \(Availability to Promise\) for Oracle Exchange](#)".

ATP (Availability to Promise) for Oracle Exchange

Availability to Promise (ATP) functionality allows buyers in exchanges to obtain availability information of products. The request from the buyer is sent as an XML document through a webMethods B2B server to the supplier. The supplier then processes the request and sends the necessary information back to the buyer as an XML document.

ATP is implemented using a combination of Java and webMethods on the following four-tier client-server architecture.

Browser (Client) <—> Exchange Server <—> webMethods B2B Server <—> Supplier Server

The Java classes:

- ATPDataService.java
- ATPItem.java
- ATPRecord.java
- ATPService.java
- ATPSupplierInfo.java
- ATPThreadService.java*
- BuildXML.java

*ATPThreadService.java is generated by webMethods.

The webMethods Services

ATP Package includes:

```
ATPRequest Interface
----> ATPThreadService (Java Service)
----> httpPost (Flow Service)
----> httpTimer (Flow Service)
```

Exchange - Supplier XML

The following is the XML document that is sent from Exchange to supplier:

Control Section:

```
Sender: Supplier information is available in this section.
Logical Id: is the supplier Id.
Auth Id: is the supplier name.
```

Data Section:

User Area:

Name1: Exchange Id.

Name2: Exchange Name.

```

<?xml version = '1.0' standalone = 'no'?>
<GET_PRODAVAIL_002>
  <CNTROLAREA>
    <BSR>
      <VERB>
        <![CDATA[GET]]>
      </VERB>
      <NOUN>
        <![CDATA[PRODAVAIL]]>
      </NOUN>
      <REVISION>
        <![CDATA[002]]>
      </REVISION>
    </BSR>
    <SENDER>
      <LOGICALID>
        <![CDATA[8821]]>
      </LOGICALID>
      <COMPONENT>
        <![CDATA[SALES]]>
      </COMPONENT>
      <TASK>
        <![CDATA[ATP]]>
      </TASK>
      <REFERENCEID>
        <![CDATA[786957]]>
      </REFERENCEID>
      <CONFIRMATION>
        <![CDATA[1]]>
      </CONFIRMATION>
      <LANGUAGE>
        <![CDATA[EN]]>
      </LANGUAGE>
      <CODEPAGE>
        <![CDATA[CPXML]]>
      </CODEPAGE>
      <AUTHID>
        <![CDATA[CHRIS]]>
      </AUTHID>
    </SENDER>
  </CNTROLAREA>
</GET_PRODAVAIL_002>

```

```

<DATETIME qualifier='CREATION'>
  <YEAR>
    <![CDATA[1999]]>
  </YEAR>
  <MONTH>
    <![CDATA[01]]>
  </MONTH>
  <DAY>
    <![CDATA[01]]>
  </DAY>
  <HOUR>
    <![CDATA[00]]>
  </HOUR>
  <MINUTE>
    <![CDATA[00]]>
  </MINUTE>
  <SECOND>
    <![CDATA[00]]>
  </SECOND>
  <SUBSECOND>
    <![CDATA[0000]]>
  </SUBSECOND>
  <TIMEZONE>
    <![CDATA[-0600]]>
  </TIMEZONE>
</DATETIME>
</CONTROLAREA>
<DATAAREA>
  <GET_PRODAVAIL>
    <PRODAVAIL returndata='1'>
      <DATETIME qualifier='REQUIRED'>
        <YEAR>
          <![CDATA[2000]]>
        </YEAR>
        <MONTH>
          <![CDATA[6]]>
        </MONTH>
        <DAY>
          <![CDATA[6]]>
        </DAY>
        <HOUR>
          <![CDATA[15]]>
        </HOUR>
        <MINUTE>
          <![CDATA[32]]>
        </MINUTE>
      </DATETIME>
    </PRODAVAIL>
  </GET_PRODAVAIL>
</DATAAREA>

```

```
</MINUTE>
<SECOND>
  <![CDATA[21]]>
</SECOND>
<SUBSECOND>
  <![CDATA[0000]]>
</SUBSECOND>
<TIMEZONE>
  <![CDATA[-8]]>
</TIMEZONE>
</DATETIME>
<QUANTITY qualifier='ORDERED'>
  <VALUE>
    <![CDATA[1]]>
  </VALUE>
  <NUMOFDEC>
    <![CDATA[0]]>
  </NUMOFDEC>
  <SIGN>
    <![CDATA[+]]>
  </SIGN>
  <UOM>
    <![CDATA[Each]]>
  </UOM>
</QUANTITY>
<ITEM>
  <![CDATA[652EGBA002]]>
</ITEM>
<SITELEVEL index='1'>
  <![CDATA[200]]>
</SITELEVEL>
<USERAREA>
<NAME index='1'>
  <![CDATA[2]]>
</NAME>
<NAME index='2'>
  <![CDATA[auto-xchange]]>
</NAME>
  </USERAREA>
  </PRODAVAIL>
  </GET_PRODAVAIL>
</DATAAREA>
</GET_PRODAVAIL_002>
```


XML Messaging Services

Oracle can send and receive key OAG compliant XML documents with other leading ERP vendors using XML Messaging Services.

Oracle XML Messaging Services is a tool that enables the production and consumption of valid, well-formed XML messages between Oracle e-Business suite and trading partners. XML Messaging Services enables application interoperability and integration supporting enterprise integration requirements driven by Business-to-Business (B2B) and Application to Application (A2A) integration requirements. XML Messaging Services is the core technology used for sending and receiving OAG compliant XML documents to trading partners in Oracle Exchange.

XML Messaging Services features include the following:

- Provides a single, consistent XML-based application integration tool
- Provides a repository based, design time user interface for message mapping and creation and a run-time engine for message processing
- Supports event driven message processing to parallel enterprise and trading partner business processes
- Supports an open, standards independent message development approach
- Provides pre-built XML messages conforming to the OAGI specifications
- Provides message validation based on a specified DTD, file, or XML Schema
- Supports rule-based exception processing
- Integration to Oracle Advanced Queueing (AQ) for placing outgoing and extracting incoming messages

XML Message Designer and Runtime Execution Engine

The Message Designer is a Java user interface that allows quick and easy creation of XML Messaging Maps which can be loaded into the XML Messaging Services repository.

Generating XML that Conforms to New Schema

You can use XSQL Servlet to produce an XML document with a structure which reflects your database schema. As you define or adopt XML schemas for various categories of data (such as timeseries ML, News ML, FpML) you also need to be able to generate XML that conforms to these schemas.

How can you generate standard XML industry schemas?

You can use the following methods to help you generate XML industry schemas:

- XML Messaging Services. This is part of Oracle Exchange and is not a stand alone component.
- XSLT stylesheets. These are useful for structural transformation, but not so good for content, such as unique ids, xrefs, and so on. Various tools allow you to write stylesheets, but they are all not much more than enhanced editors. Generation, repositories, and so on are not readily available.
- Specialized transformation tools, such as Mercator and Constellar. These costly tools have drag and drop capability and support wizard-based development.

Introducing Oracle XML Gateway

This chapter describes the following sections:

- [What is XML Gateway?](#)
- [Oracle XML Gateway Services](#)
- [Oracle XML Gateway Architecture](#)
- [XML Gateway Services - Message Designer](#)
- [XML Gateway Services - Message Set Up](#)
- [XML Gateway Services - Execution Engine](#)
- [A Word About XML Standards](#)

What is XML Gateway?

With Release 11i.4 of the Oracle e-Business Suite, Oracle XML Gateway emerges as a key component of Oracle's application integration framework. XML Gateway is a set of services that allows for easy integration with the Oracle e-Business Suite to create and consume XML messages triggered by business events. It integrates with Oracle Advanced Queuing to enqueue/dequeue a message which is then transmitted to/from the business partner via any message transport agent.

Oracle XML Gateway Services

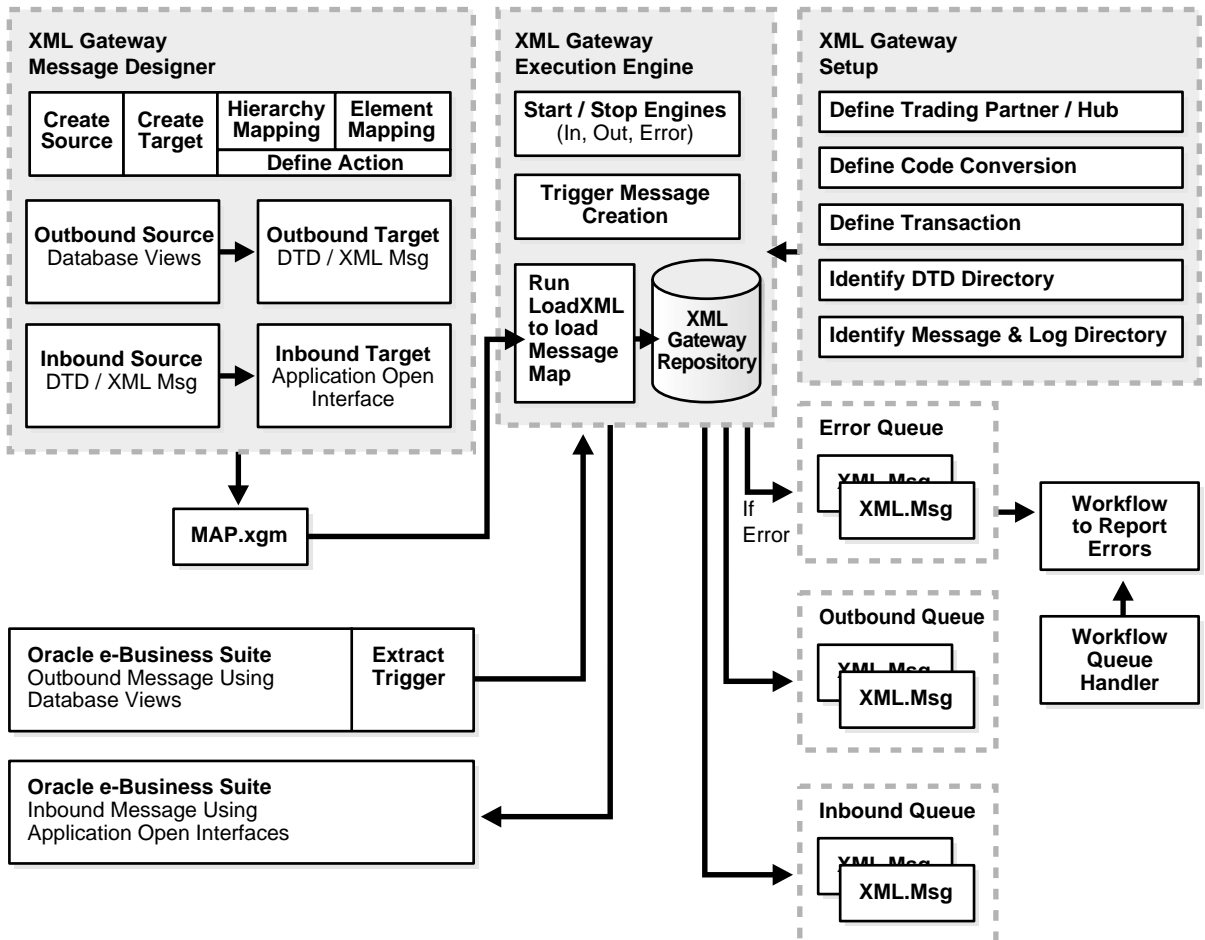
Oracle XML Gateway provides the following services:

- Wizard-guided, GUI-oriented, repository-based Message Designer to define data source and targets, create hierarchy and element maps plus define actions for data transformation and process control.
- Robust execution engine integrated with the Oracle e-Business Suite to create and consume XML messages based on a business event. Messages are created or consumed based on the message map (associated with the trading partner) stored in the XML Gateway repository.
- Flexible trading partner definition to accommodate a hub, all trading partners exchanging on a hub, or a specific business partner.
- Site level trading partner directory service to enable a message, identify a message map, and identify the communication protocol.
- Flexible message set up to define code conversion values and transaction names between the sender and recipient.
- Integration with the XML Parser to ensure that XML Gateway creates or consumes well-formed and valid (if DTD available) XML messages.
- Active notification via Oracle Workflow to report errors detected by the XML Gateway Execution Engine, Oracle Advanced Queuing or a transport agent.
- Integration with Oracle Advanced Queuing to enqueue/dequeue outbound or inbound XML messages. In addition, Oracle AQ is used to enqueue/dequeue error messages to support active error notification.
- Integration of Oracle Advanced Queuing with Oracle Workflow to deliver/receive XML messages.

Oracle XML Gateway Architecture

The services supported by Oracle XML Gateway are grouped into three major components: Message Designer, Message Set Up, and Execution Engine. Figure 17-1 shows the relationship of these three components:

Figure 17-1 XML Gateway Architecture



XML Gateway components are integrated with the following Oracle tools and technologies to provide a complete solution:

- XML Parser to validate XML message is well-formed and valid (if DTD or XML Schema is available)
- Oracle Workflow to report processing errors
- Oracle Advanced Queuing to en-queue/de-queue an XML message
- Oracle Workflow and transport agent to deliver/receive an XML message

XML Gateway Services - Message Designer

The XML Gateway Message Designer is a wizard-guided, GUI-oriented, repository-based tool that allows you to do the following:

- **Create Data Source.** Every message must get its data from some data source to create the target message meaningful to the recipient. The possible data sources supported by the Message Designer are database tables, database views, DTD or sample XML message. The common data source for outbound messages is database views (new or existing) and the relevant view columns required by the message. The common data source for inbound messages is a DTD from any XML standards body or a sample XML message.

Selecting a sample XML message as a data source is a wise choice if you are transitioning from an existing implementation/legacy system to the Oracle e-Business Suite. For each source column identified, you can use Message Designer to perform the following:

- Define document levels if source is a DTD or XML Schema as they (DTDs) do not have the concept of document level
- Set default values
- Enable code conversion
- Indicate whether column is mandatory or not
- Add/delete sibling (same hierarchy) or child (next level of detail) node as necessary
- **Create Data Target.** As with data sources, every message must have a data target. The possible data targets supported by the Message Designer are identical to what's supported for data sources. The common data target for outbound messages is a DTD from any XML standards body or a sample XML message. The common data target for inbound messages is the Application Open Interface tables (new or existing) and the relevant columns required by the message. Selecting a sample XML message as a data target is a wise choice if

you are transitioning from an existing implementation/legacy system to the Oracle e-Business Suite. For each target column identified, you can use Message Designer to perform the following:

- Define document levels if target is a DTD as DTDs do not have the concept of document level
- Set default values
- Indicate whether column is mandatory or not
- Add/delete sibling (same hierarchy) or child (next level of detail) node as necessary

Code conversion is enabled at the data source only as you are converting the source value to the value required by the recipient.

- **Perform Hierarchy Mapping.** Once the data source and data target is created, use the Message Designer's hierarchy mapping to relate the source data structure to the target data structure. If the document levels between the source and target are different, use the Message Designer to expand or collapse document levels as necessary.
- **Perform Element Mapping.** Once the source and target are created and the hierarchy between the source and target is defined, use the Message Designer to map the source data element to the target data element. The Message Designer displays the data source on the left pane and the data target on the right pane. A simple drag and drop between the source and target data element creates a map relationship. The source data element name is noted next to the target data element name to identify the map relationship.
- **Define Actions.** As part of the hierarchy or element mapping process, you can use the Message Designer to define actions for data transformation or process control. An action may be defined as follows:
 - At the source or target
 - Applied at the data element, document, or root level
 - Applied before, during or after the message is created or consumed

An action may be based on a pre-defined condition. If no condition is defined, the action will always be applied.

[Table 17-1](#) lists the actions supported by XML Gateway.

Table 17–1 Actions Supported by XML Gateway

Action Category	Action Description
Assignments	Create global variable Assign value from another variable
Math Functions	Add Subtract Multiple Divide
String Functions	Sub-string Concatenate
Database Functions	Assign next sequence value Append where clause Insert into database table
Procedure Call	Execute procedure with send and return parameters
Function Call	Execute function and assign function return value
XSLT Transformation	Execute procedure to perform XSLT transformation
OAG Standard Conversions	Convert Oracle date to OAG date format Convert Oracle operational amount to OAG operational amount format Convert Oracle quantity to OAG quantity format Convert Oracle amount to OAG amount format Convert OAG date to Oracle date format Convert OAG operational amount to Oracle operational amount Convert OAG quantity to Oracle quantity format Convert OAG amount to Oracle amount format
Return Error Code to Sender	Return error code and error message to sender

Table 17–1 Actions Supported by XML Gateway (Cont.)

Action Category	Action Description
Get Global Variable Value	Get global variable value <ul style="list-style-type: none"> ▪ DOCUMENT_ID ▪ RETURN_CODE ▪ RETURN_MESG ▪ SENDER_TP_ID ▪ RECEIVER_TP_ID ▪ CODE_CONVERSION
Other	Exit program

The most common actions are the OAG standard conversions to convert Oracle's representation for date, operational amount, quantity, and amount values to the OAG format and vice versa. The ability to inquire on the status of the Execution Engine allows flexible process control based on the severity of an error. For serious errors, the process may be aborted with error messages returned to the sender via an Oracle Workflow process. The convenience of calling out to existing procedures or functions enables tight integration with the Oracle e-Business Suite. Once the message map is defined, it is loaded into the XML Gateway repository for use by the Execution Engine to create outbound or consume inbound XML messages.

XML Gateway Services - Message Set Up

To implement a message with a trading partner, use XML Gateway Services message set up to define the trading partner or hub, code conversion values, and transaction name cross references. In addition, you can identify where on the file system to store the DTDs, XML messages and process LOG files.

- **Define Trading Partner/Hub.** E.-Business may be conducted directly with a business partner commonly known as a trading partner or via a hub such as Exchange where many buyers and sellers converge to conduct electronic commerce.

With Oracle XML Gateway services, you can define the hub or the actual business partner as a trading partner. If you define the hub as the trading partner, you can identify all the buyers and sellers who are conducting business on the hub as trading partners to the hub.

Included in the trading partner/hub definition is the following information:

- Trading Partner/Hub name
- Message enabled
- Message map to use for message creation or consumption
- Communication protocol - SMTP, HTTP, HTTPS and username/password as necessary
- Trading partner specific code conversion values
- **Define Code Conversion.** The Oracle XML Gateway service for code conversion allows you indicate what to convert an Oracle code to so that it is meaningful to the recipient or vice versa. Common Oracle e-Business Suite codes requiring code conversion are units of measure and currency code. Oracle XML Gateway provides a seeded master list of code conversion values which may be applied to any data element of any message. Additional code conversion values may be added to the master list if the seeded list is insufficient. In addition, you may define a trading partner specific code conversion value which will be applied for that trading partner only.
- **Define Transaction.** Use Oracle XML Gateway to define a cross reference between the Oracle transaction name and a transaction name meaningful to the recipient. For the pre-built messages delivered with the Oracle e-Business Suite, the cross referenced name is the verb/noun (i.e. Process PO or Show Delivery) combination defined by the OAG Business Object Document (OAG BOD).
- **Identify DTD Directory.** Use Oracle XML Gateway to identify a directory on the file system to store the DTD used to implement the message. The XML Gateway Execution Engine and XML Parser will use the DTDs stored in this directory to validate all outgoing and incoming messages to ensure that they are well-formed and valid.
- **Identify XML Message and Process Log Directory.** Use Oracle XML Gateway to identify a directory on the file system for the XML Gateway Execution Engine to store a copy of the XML message and it's associated process log file. Both the XML message and process log file may be archived or used for trouble shooting.

XML Gateway Services - Execution Engine

The XML Gateway Execution Engine interfaces with the Oracle e-Business Suite, XML Gateway Set Ups and Oracle Advanced Queuing to perform the following functions:

- Start/Stop Engines associated with the inbound, outbound, and error queues
- Allow users to manually trigger message creation if message creation is not trigger by a business event in the Oracle e-Business Suite
- Interface with Oracle e-Business Suite to produce an outbound message
- Validate Trading Partner/Hub. If the Trading Partner is not defined or the document is not defined for the Trading Partner, no message will be produced.
- Get Message Map from Repository. If the message map associated with the Trading Partner is not available in the XML Gateway repository, no message will be created.
- Gather Application Data. If the Trading Partner is valid and the message map exist in the repository, the XML Gateway Execution Engine gathers the application data from the Oracle e-Business Suite using the database view and columns identified in the message map.
- Apply Code Conversion. Apply code conversion for source columns enabled for code conversion
- Apply Actions. Apply actions where defined (may be document or element level)
- Create XML Message. Create XML message using the message map and the application data
- Validate Message via XML Parser. Use the XML Parser to validate the newly created message to ensure that it is well-formed and valid. A poorly formed or invalid message (based on DTD stored in DTD directory) will not be en-queued onto the Outbound Queue.
- Enqueue Message to Outbound Queue. Enqueue well-formed and valid message onto the Outbound Queue to be picked up by the transport agent for delivery to the trading partner.
- Interface with Oracle e-Business Suite to consume an inbound message
- Dequeue Message from Inbound Queue

- **Validate Message via XML Parser.** Use the XML Parser to validate the inbound message to determine if it is well-formed and valid (based on DTD stored in DTD directory) before proceeding further.
- **Validate Trading Partner/Hub.** If the inbound message is both well-formed and valid, the Execution Engine proceeds to validate that the Trading Partner and document are defined. If the Trading Partner is not defined or the document is not defined for the Trading Partner, the message cannot be processed further.
- **Get Message Map from Repository.** If the message map associated with the Trading Partner is not available in the XML Gateway repository, the message cannot be processed further.
- **Apply code conversion for source columns enabled for code conversion**
- **Apply actions where defined (may be document or element level) including inserting data into the Application Open Interface tables and then finally executing the Open Interface API to populate the base application tables.**
- **Detect and Report Processing Errors.** Errors may be detected by the Oracle XML Gateway Execution Engine, Oracle Advanced Queuing, or a transport agent. Information regarding the error is en-queued onto the Error Queue. A notification is sent via Oracle Workflow to notify the trading partner regarding data errors or the XML Gateway system administrator regarding system/process errors.

In addition, for system/process errors, a copy of the XML message is placed in the XML message directory for use in trouble shooting the reported error. For trading partner related data errors, the trading partner can refer to their copy of the XML message. The XML Gateway listeners are actively polling for messages and will begin processing once it detects that something has arrived on the transaction queue. The XML Gateway Execution Engine will take the document information from the transaction queue and begin the process of creating or consuming an XML message as described above.

A Word About XML Standards

Many standards bodies (for example, EbXML, Rosettanet, SOAP, iFX) exist with published Document Type Definitions (DTD) each claiming to be better than the other. Some standards are strong at managing the message content while others excel at managing both the message content and its related processes.

As a provider of software to support all industries, Oracle has chosen to align with Open Application Group's (OAG) XML standards for broad based message

implementation. OAG is also the standard most widely adopted by the Oracle customer base. All Oracle pre-built messages delivered with the Oracle e-Business Suite will be based on the OAG standards. However, all pre-built messages may be re-mapped to any standard of choice using the XML Gateway Message Designer provided a DTD or XML Schema Definition is available.

Part V

OracleAS Dynamic Services (DS) and Oracle Syndication Server (OSS)

Part V contains the following chapters:

- [Chapter 18, "Using OracleAS Dynamic Services and XML"](#)
- [Chapter 19, "Oracle Syndication Server \(OSS\) and XML"](#)

Using OracleAS Dynamic Services and XML

OracleAS Dynamic Services (Dynamic Services or DS) enables developers to take advantage of Web application functionality and content, as services to increase productivity. DS adds web services deployment and management capability to Oracle. Developers can use Oracle9i Dynamic Services to compose, catalog, manage, and personalize web services according to user roles, protocols and delivery devices.

This chapter describes the following sections:

- [Introducing OracleAS Dynamic Services](#)
- [What is Needed to Run OracleAS Dynamic Services?](#)
- [Dynamic Services \(DS\) Architecture Overview](#)
- [Dynamic Services \(DS\) Implementation Overview \(Java, PL/SQL, HTTP/Java\)](#)
- [Dynamic Services Features](#)
- [Dynamic Services Integrates with Other Oracle Products](#)
- [How Service Consumers Use Dynamic Services](#)
- [Developing Services For Dynamic Services](#)
- [Oracle Syndication Server \(OSS\)](#)
- [Dynamic Services Consumer Application: Stock Portfolio Example](#)
- [Frequently Asked Questions \(FAQs\): Dynamic Services](#)

Introducing OracleAS Dynamic Services

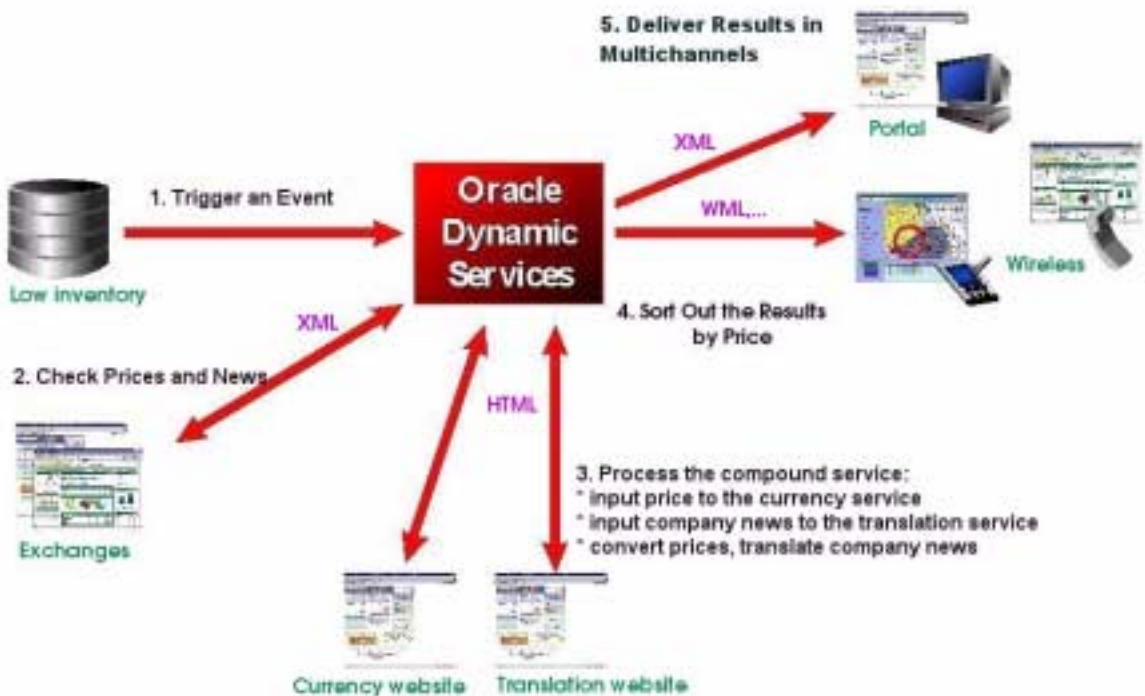
OracleAS Dynamic Services (DS) is a Java-based programmatic framework for incorporating, managing, and deploying Internet and Intranet services. It uses the Internet as the information source. It facilitates the rapid incorporation of services from Web sites, local databases, and proprietary systems into applications.

For example, an online financial portfolio application can use DS to integrate Internet financial services, such as stock quotes and exchange rates, from different resource providers to calculate the current value of a portfolio in foreign currency. See also [Figure 18-1](#) and "[Dynamic Services Consumer Application: Stock Portfolio Example](#)" on page 18-18. DS handles dynamic business models with no degradation in service quality. A DS "service" provides access to information or application functionality through standard Internet protocols, such as, HTTP, JDBC, or SOAP. A "service" can also aggregate other DS services to form a compound service with a specific execution flow. It can include transformations and conditional logic and is accessible through a uniform interface.

[Figure 18-1](#) illustrates a typical Dynamic Services (DS) scenario. This is a workflow of how an application can use DS services to retrieve and manage the data. You simply need to determine the semantics and output formats needed:

1. The database notes that the inventory is getting low for a certain part in your inventory. This triggers an event that tells you a specific part is low.
2. A DS service is then invoked that access a supplier and invokes the service that accesses the information. The supplier ABC's site could be in a foreign currency or different format. Data about the supplier is logged as well as a catalog of parts available.
3. Another service is invoked that translates the language and currency to the desired language and currency of your Inventory Application.
4. The inventory application can also be connected to other applications. In steps 4 and 5 the results are delivered in multiple channels.
5. The resulting data is formatted according to the devices target user's devices. The DS service can also render the results in any format that the service has been equipped for using stylesheets. The data is transferred mostly in XML.

Figure 18–1 Typical Dynamic Services Scenario



How Dynamic Services (DS) Helps Developers

DS meets the design criteria needed by application developers in that with DS, developers have:

- A single programmatic framework for service access, management, execution, and delivery, based on the following features:
 - **Access:** Uniform, XML-based access to diverse sources through different protocols.
 - **Management:** Centralized management of services, sessions, caching policies, and events.
 - **Execution:** Advanced execution modules to enable service aggregation and failover.

- **Delivery:** Multi-channel delivery to different output formats, devices, and user groups
- Each business can define its own internal service interfaces, without forcing partners to standardize. In other words, if you have partners with various interfaces, languages, currencies, and so on, they will not have to make any changes in order to exchange data with you.

For Further Information

For further information about OracleAS Dynamic Services:

See:

- *Oracle Directory Service Integration and Deployment Guide*
- *Oracle Dynamic Services User's and Administrator's Guide*
- http://otn.oracle.com/products/dynamic_services/index.htm

What is Needed to Run OracleAS Dynamic Services?

To run OracleAS Dynamic Services you need the following:

- Oracle9i Enterprise Edition, OracleAS
- Any platform that has Java2-compliant (JDK 1.2.2 or later) JVM installed, including Oracle8i JVM or higher. This is because the Dynamic Services engine is implemented in Java.
- JDK 1.2.2 or higher. <JAVA2_HOME> is the installation directory of the JDK 1.2.2 or higher, distribution.

Ensure that you have at least a full (typical) installation of Oracle.

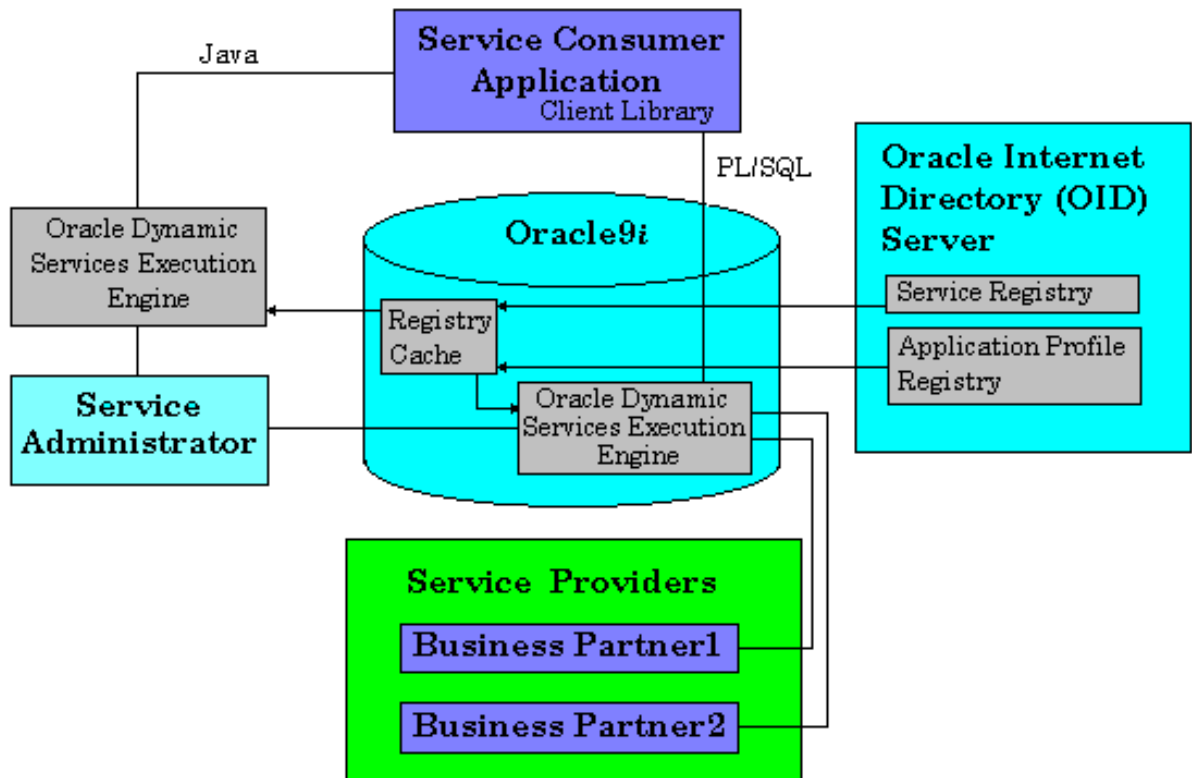
Dynamic Services (DS) Architecture Overview

Figure 18-2 shows an overview of Oracle Dynamic Services (DS) architecture. It shows the following:

- How Service Providers (business partners and application developers) provide services that Service Administrators register in the Service Registry using the DSAdmin utility (Service Administrator).
- Application developers can create applications using application profiles that service administrators register in the Application Profile Registry. The registry

is an Oracle Internet Directory (OID) Lightweight Directory Access Protocol (LDAP) server whose contents are also mirrored in the Oracle database for performance optimization.

Figure 18-2 Oracle Dynamic Services (DS) Architecture



Dynamic Services can be deployed in the following ways:

- Using PL/SQL interfaces when deployed within the Oracle JVM (see [Figure 18-4](#))
- Using Java interfaces, when deployed on a local machine hosting the application (thick client library) (see [Figure 18-3](#))

- Using remote Java interfaces when deployed as a middle-tier Java engine behind a Java servlet with which the application can communicate through Dynamic Services thin client library (see [Figure 18-5](#))

Dynamic Services (DS) Implementation Overview

OracleAS Dynamic Services (DS) currently offers deployment modes:

- **Java deployment** view (see "[Dynamic Services Java Deployment](#)" on page 18-7).
- **PL/SQL deployment** view (see "[Dynamic Services PL/SQL Deployment](#)" on page 18-8).
- **Java (HTTP/Java Messaging Services (JMS)) Deployment** view (see "[Dynamic Services Java HTTP/Java Messaging Services \(JMS\) Deployment](#)" on page 18-9).

Main DS Components

The following lists the DS main components for each of these deployment modes.

- **Dynamic Services Engine.** The Dynamic Services engine can be deployed as any of the following three engine types:
 - A Java engine running on the machine hosting the application (thick client library) (see [Figure 18-3](#)).
 - A middle-tier Java engine behind a Java servlet (see [Figure 18-5](#)).
 - An engine running within Oracle JVM (see [Figure 18-4](#)).

You can switch from one environment to another without recompiling or even restarting your application. This gives you a way to try out the various options.

- **DS Service and Application Profile Registries.** The Service Registry and Application Profile Registry are used as directories in Oracle Internet Directory (OID) server. The access control list of OID is used for access control, allowing service administrators to choose the services visible to a particular service consumer application.
- **Communication Between Service Consumer Applications and Dynamic Services Engine.** Communication between the Dynamic Services engine and the service consumer applications is handled by the Dynamic Services client library. By registering a Dynamic Services driver, a service consumer application can dynamically change the underlying communication protocol used by the client library to communicate with the Dynamic Services engine.

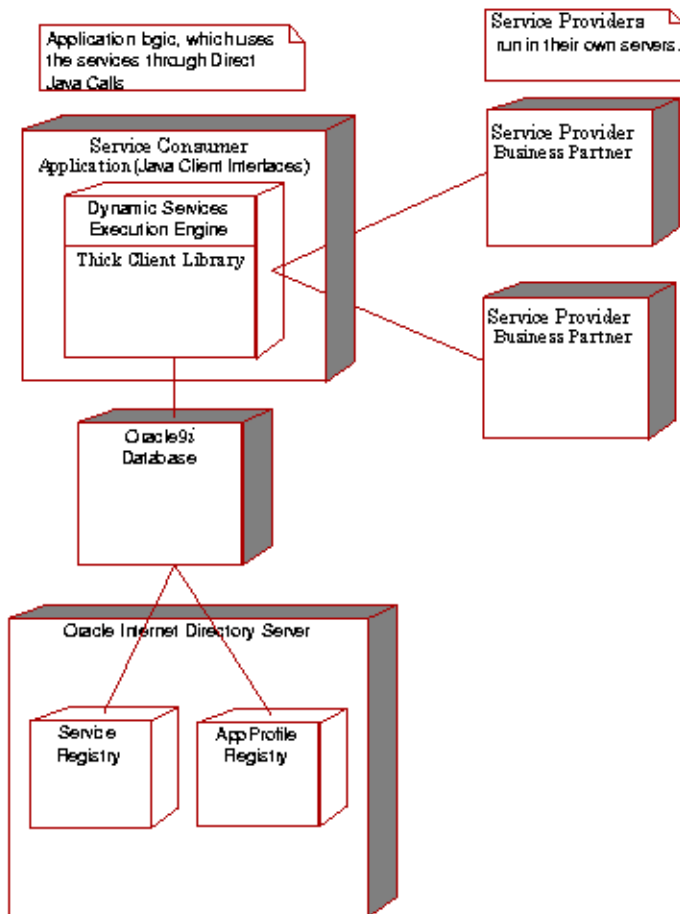
Supported communication protocols include HTTP, AQ/JMS, and direct Java access.

Dynamic Services Java Deployment

[Figure 18-3](#) shows the Dynamic Services Java deployment view. Oracle serves as a registry cache, communicating with the OID Lightweight Directory Access Protocol (LDAP) server hosting the registries. Service Consumer Application contains application logic that uses the services through direct Java calls.

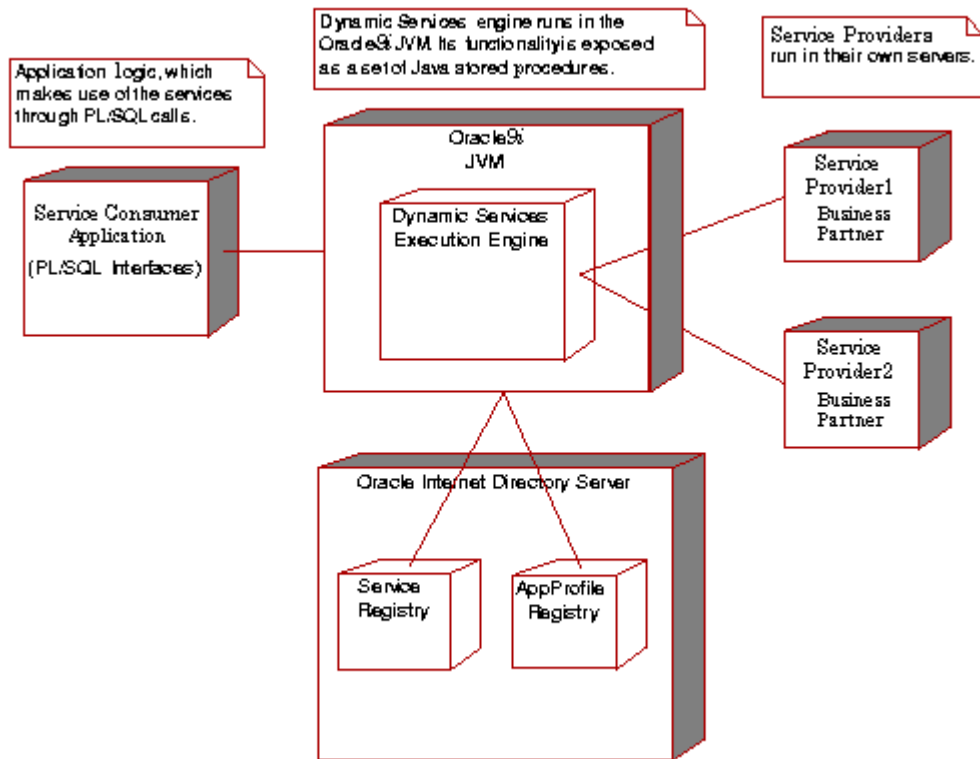
Here, Service Consumer Application uses the DS thick client library, that contains the Dynamic Services Execution Engine. Service providers run in their own servers.

Figure 18–3 Oracle Dynamic Services: Java Deployment



Dynamic Services PL/SQL Deployment

Figure 18–4 shows Oracle Dynamic Services PL/SQL deployment. The Dynamic Services engine runs in the Oracle JVM, with its functions exposed as a set of Java stored procedures. Oracle database serves as a registry cache. It communicates with Oracle Internet Directory LDAP server which hosts the registries. The Service Consumer Application's logic makes use of the services through PL/SQL calls. Service Providers run in their own servers.

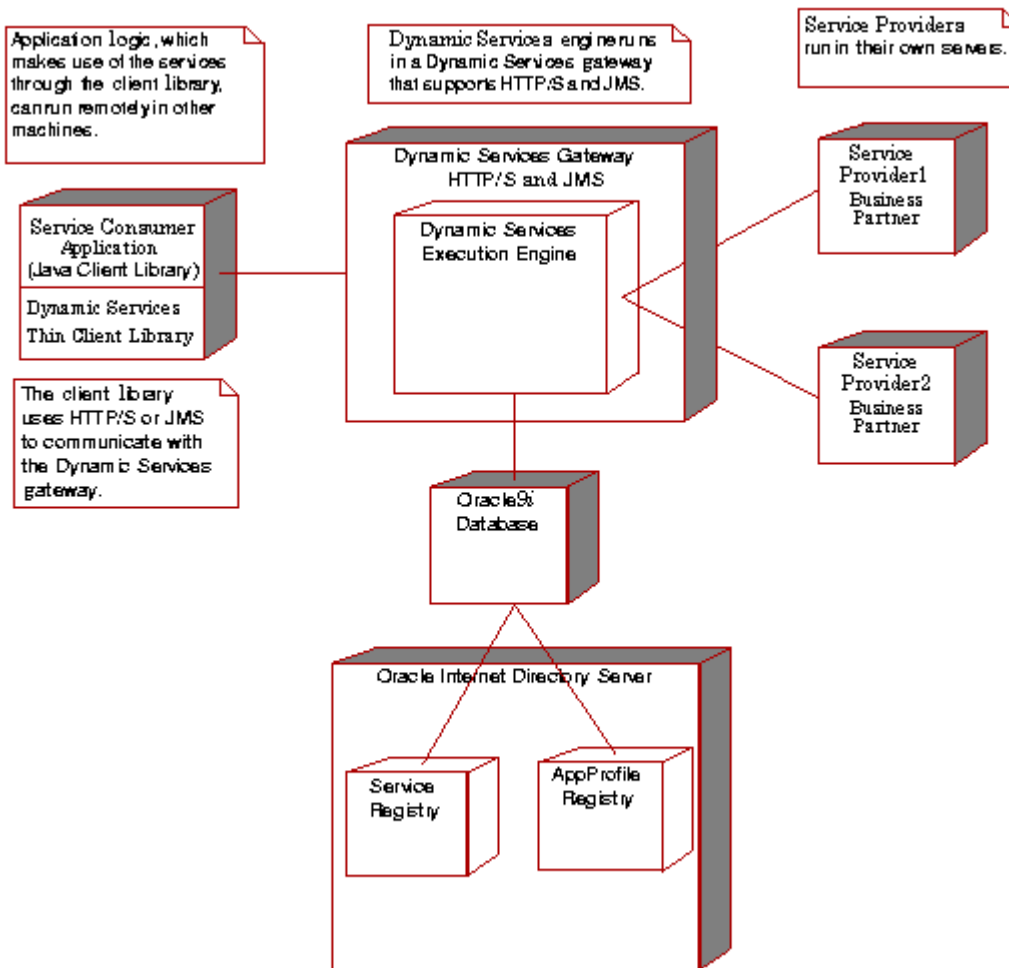
Figure 18–4 Oracle Dynamic Services: PL/SQL Deployment

Dynamic Services Java HTTP/Java Messaging Services (JMS) Deployment

Figure 18–5 shows an Oracle Dynamic Services Java (HTTP/JMS) deployment view. The Dynamic Services engine running in a Dynamic Services gateway (middle tier) supports HTTP, HTTPS, and JMS communication protocols. Oracle database serves as a registry cache, communicating with the Oracle Internet Directory LDAP server hosting the registries. The Service Consumer Application's logic, makes use of the services through Dynamic Services' thin Java client library, and executes services remotely in other systems. Here, service execution requests are forwarded to the Dynamic Services gateway, which executes the service and returns a response.

Communication between the service consumer application and the gateway is handled by the Dynamic Services thin client library.

Figure 18–5 Oracle Dynamic Services Java (HTTP/JMS) Deployment



For example, in asynchronous deployment communications (JMS), the DSJMSDriver can allow for asynchronous access to services, in the form of a JMS

daemon (using a Dynamic Services gateway). The driver allows request submission asynchronously to an AQ/JMS queue in a remote database. The driver assumes the existence of this JMS daemon running somewhere that listens asynchronously to the same queue where requests are being submitted.

The JMS daemon takes on the role of the Dynamic Services engine. It processes the request, generates a response, and submits the response to another queue that the DSJMSDriver listens to asynchronously. On the service consumer application side, therefore, listeners can be registered to be informed when the response is returned.

See Also:

- *Oracle Directory Service Integration and Deployment Guide*
- *Oracle Dynamic Services User's and Administrator's Guide*

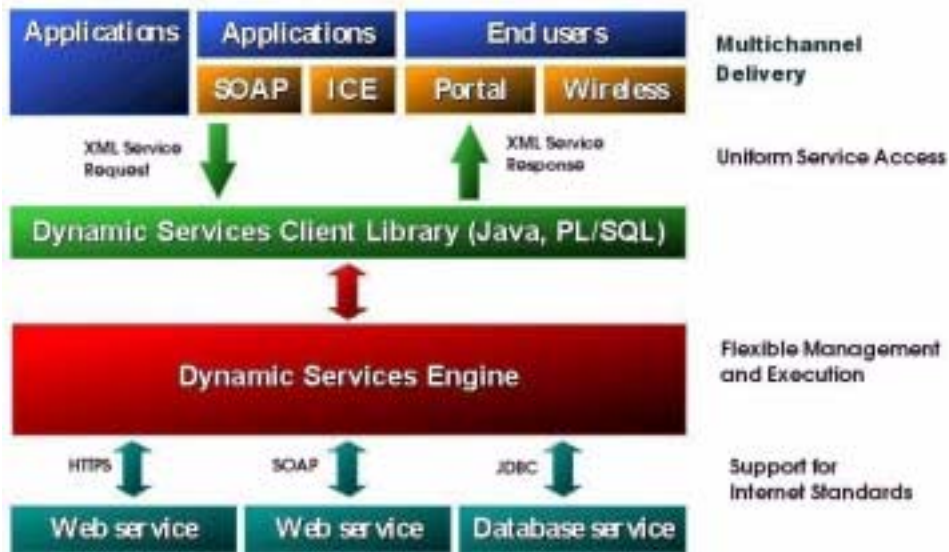
for more information about Dynamic Services' Java, PL/SQL, and JMS deployment.

Multiple Channel Capabilities of DS

[Figure 18-6](#) illustrates the multiple channels through which DS can be delivered. These include:

- Oracle Portal
- Oracle Application Server Wireless Edition (OracleAS WE)
- ICE through the Oracle Syndication Server (OSS)
- SOAP through the Oracle Dynamic Services SOAP Listener

Figure 18–6 Dynamic Services Overview



Dynamic Services Features

Dynamic Services (DS) features include the following:

Service Management and Administration

Dynamic Services' (DS) service management and administration features include: DS business relationship management can handle such business tasks as:

- Specifying service policies, priorities
- Working with partner security or authentication models
- Tracking usage and perform billing through events
- Maintain copyright, logo, and other provider information in service descriptor

In forthcoming releases, DS will be managed and administered through Oracle Enterprise Manager.

Service Discovery

Dynamic Services' service discovery feature supports run-time Intranet Discovery (LDAP). For example, service descriptors can be stored in Oracle Internet Directory (OID) for security, centralized management, and LDAP lookup. They can be accessed from mirrored Oracle instances for improved runtime performance.

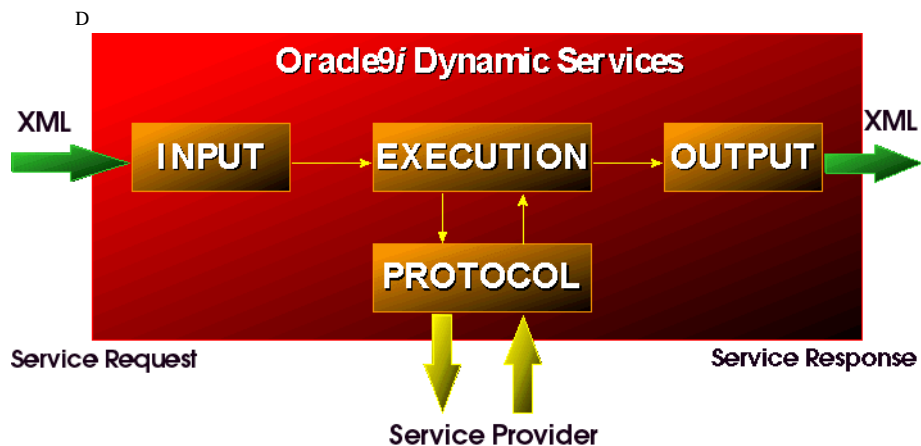
Service Execution

Dynamic Services service execution features include the following:

- Executing a service. [Figure 18-7](#) shows an overview of this.
- Advanced Execution Modules:
 - Failover
 - Compound
 - Conditional services
- Events/triggers

The following sections and diagrams describe these DS service execution features.

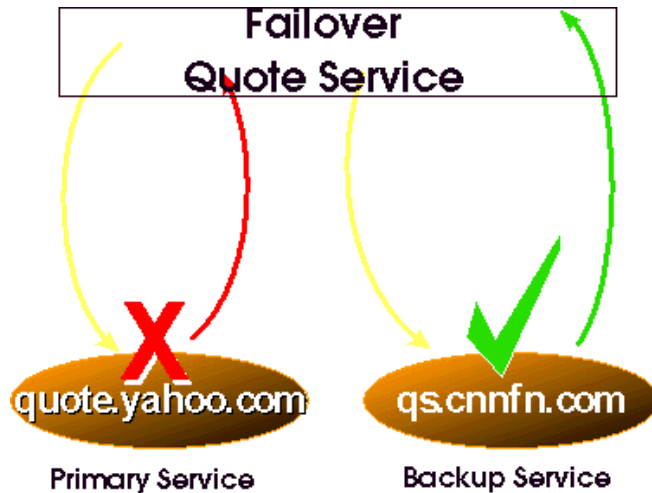
Figure 18-7 DS: Executing a Service — Overview



Failover Services

DS Failover services is a prioritized list of backup equivalent services. [Figure 18-8](#) shows an example of failover services used in a stock quote application.

Figure 18-8 Dynamic Services: Failover Services Example

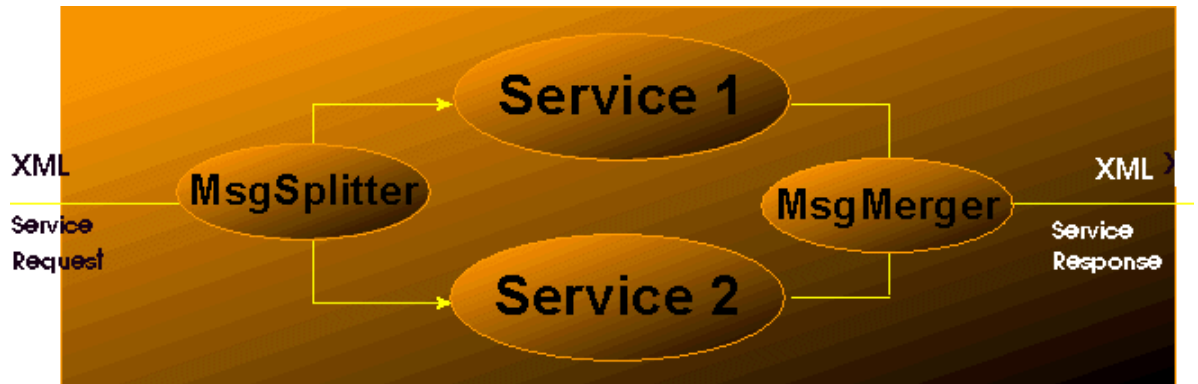


Compound Services

DS allows you to aggregate services and specify which services you need executed in parallel and which services you need executed in series.

For example, [Figure 18-9](#) shows how you can take one or more services, here Service 1 and Service 2, and combine the results by performing operations (here merging or splitting) on them to generate a single result. You can also take the output of one service and input this into another service.

Figure 18–9 *Dynamic Services: Compound Services*

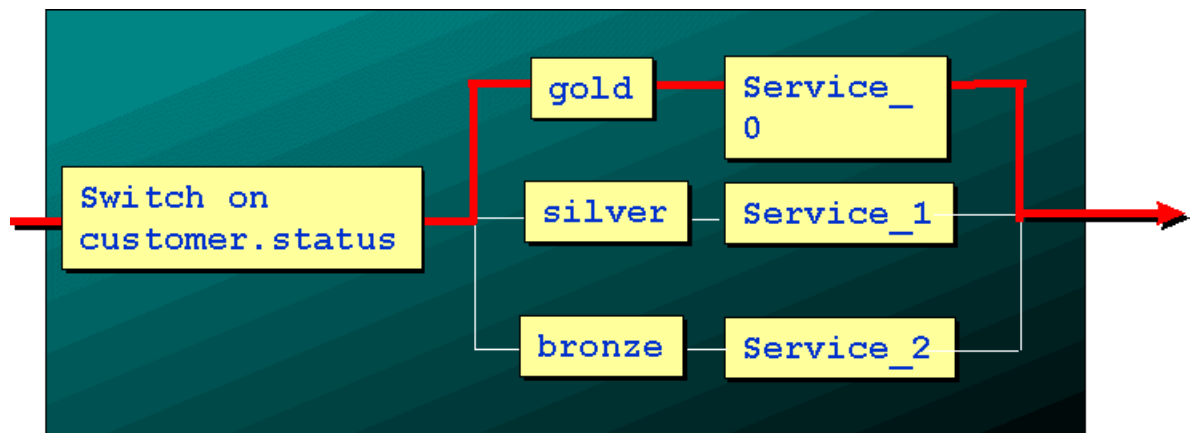


Conditional Services

Dynamic Services can execute services according to business requirements. Requirements can be based on service request or user profile properties.

Figure 18–10 shows how you can use DS to switch on a specific service(s) for 'gold' customers.

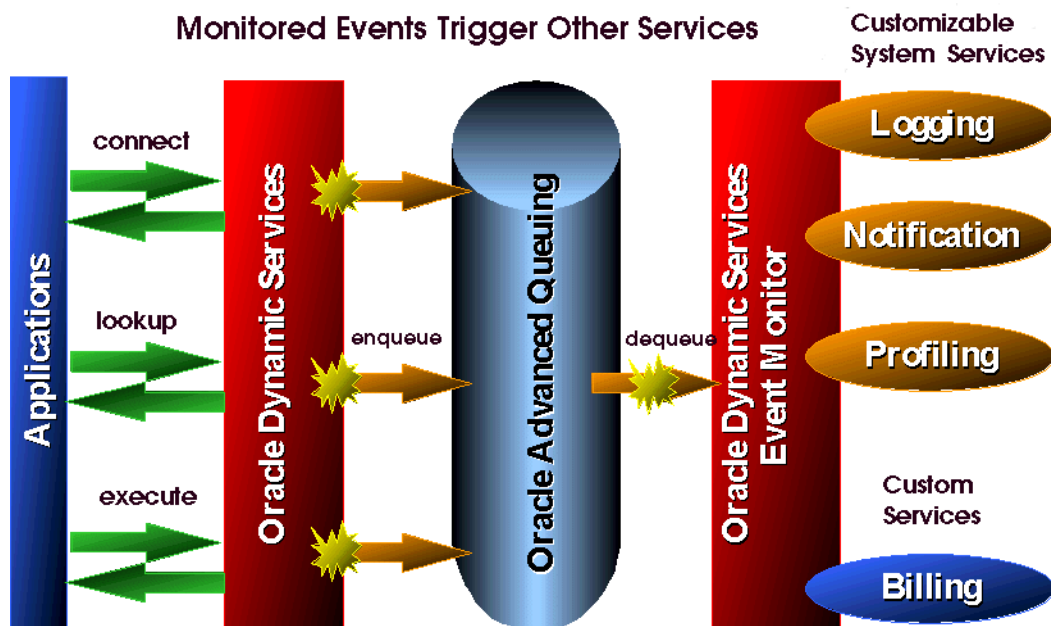
Figure 18–10 *DS: Conditional Services*



Events or Triggers

Through the course of executing a DS service, numerous events will be generated that can be captured by the monitoring application. This application will in turn execute other DS services, termed, “monitoring services”, depending on the events they receive. [Figure 18–11](#) illustrates an example of this and how the monitored events can trigger a variety of services, including logging, notification, profiling, and billing services.

Figure 18–11 Dynamic Services: Events or Trigger Services



Dynamic Services Integrates with Other Oracle Products

Dynamic Services (DS) integrates with other Oracle products, including:

- **JDeveloper.** You can use JDeveloper to create, debug, and deploy DS. JDeveloper facilitates application integration in Java, JSP, XSQL, and SOAP.
- **OracleAS Wireless Edition.** DS integrates with Oracle Application Server Wireless Edition (WE), previously known as Portal-To-Go. You can build an

adaptor for Oracle Application Server WE to make use of Dynamic Services in your mobile applications.

- **Oracle Portal.** You can integrate with Oracle Portal through the Oracle Portal JPDK to create a Java Web Provider that uses Dynamic Services. This exposes Dynamic Services as portlets.

Under “Portal Development Kit (PDK)”, select the PDK (“click here”) then “Integrating Technologies”. Read further information under Developing Portlets with Dynamic Services Portlet

How Service Consumers Use Dynamic Services

The Dynamic Services client library provides service consumers (application developers) with a Java application programming interface (API) that can be used to access the functions of the Dynamic Services engine.

See Also:

- *Oracle Directory Service Integration and Deployment Guide*
- *Oracle Dynamic Services User’s and Administrator’s Guide*

for client Java code examples used to create a service request for some of the sample services supplied with Oracle Dynamic Services, and executing them.

For more information, refer to the sample code in `<$ORACLE_HOME>/ds/demo/consumer` directory and to the Javadoc API (apidoc.zip) in `<$ORACLE_HOME>/ds/doc/`.

Developing Services For Dynamic Services

In Dynamic Services, a service is a component within the Internet computing model that delivers a specialized value-added function. A service is bundled into a simple service package and structured as a local directory. The following are typical tasks you will be using Dynamic Services for:

- Enabling Persistent Auditing or Event Monitor Services
- Enabling Event Logging and the event monitor
- Using the event logger monitor service
- Querying the logger events

- Modifying Cache Parameter values

Service Response Caching

The Dynamic Services engine uses the Oracle database for caching service responses. The caching policy for a given service is controlled through deployment parameters in the service descriptors. Before registering a service, the Service Administrator can review these parameters and modify them as needed. The caching parameters are defined in the `SERVICE_HEADER`, `DEPLOYMENT`, and `CACHING` elements in the service descriptor.

To change the caching parameters of a given service, you must unregister the service and register it again with the new parameter settings. Available caching parameters are:

- `MAX_AGE`: Specifies the number of seconds the service response remains valid in the cache. After the specified amount of time elapses, the cached response is discarded. When the `MAX_AGE` value is specified to be zero or less, the service response is never cached.
- `SESSION_PRIVATE`: Takes a Boolean value (`TRUE` or `FALSE`) to indicate whether cached responses for this service should be visible only within the current session, or if they should be visible to all executions. Table 7-1 shows an overview of the behavior of four possible service response cases.

See:

- *Oracle Directory Service Integration and Deployment Guide*
- *Oracle Dynamic Services User's and Administrator's Guide*

chapters 3 and 6, and Appendix C, "Descriptive Matrix of the packaged XML Schemas and Adaptors", for detailed information about developing Dynamic Services services.

Oracle Syndication Server (OSS)

OSS is an application of Dynamic Services. See [Chapter 19, "Oracle Syndication Server \(OSS\) and XML"](#).

Dynamic Services Consumer Application: Stock Portfolio Example

The Dynamic Services software package contains sample code to specifically invoke Yahoo Portfolio service through Dynamic Service engine, using Java. Portfolio

service is a service that takes stock symbols as input and give quotes as responses. This service is provided by Yahoo®.

This Dynamic Services software contains the following:

- `SampleStock.java`: Example java code to invoke a dynamic service in Java.
- `yapfl_req.xml`: Sample service request file.
- `readme.txt`

Compiling SampleStock.java

Before compiling `SampleStock.java`, include the following library in your CLASSPATH:

- Dynamic Services: `C:\Oracle\Ora81\ds\lib\ds.jar`
- Oracle XML Parser 2.0.2.9: `C:\Oracle\Ora81\ds\lib\xmlparserv2.jar`
- Oracle XMLSchema Parser 1.0: `C:\Oracle\Ora81\ds\lib\xschema.jar`
- Oracle AQ and JMS:
`C:\Oracle\Ora81\RDBMS\jlib\jmscommon.jar;C:\Oracle\Ora81\RDBMS\jlib\aqapi.jar`
- JSSE 1.0:
`C:\Oracle\Ora81\ds\lib\jcert.jar;C:\Oracle\Ora81\ds\lib\jsse.jar;C:\Oracle\Ora81\ds\lib\jnet.jar`
- LDAP JNDI:
`C:\Oracle\Ora81\ds\lib\providerutil.jar;C:\Oracle\Ora81\ds\lib\ldap.jar;C:\Oracle\Ora81\ds\lib\jndi.jar`
- ODS - XSQL 1.0.3 and dependent libraries:
- `C:\Oracle\Ora81\ds\lib\sax2.jar;C:\Oracle\Ora81\ds\lib\oraclexsql.jar;C:\Oracle\Ora81\ds\lib\xsu12.jar` where `C:\Oracle\Ora81` should be replaced with your `$ORACLE_HOME`. If there are any other libraries in red, please fix them appropriately.

Compile `SampleStock` at the command line as follows:

```
javac SampleStock.java
```

To run and test the sample, proper arguments should be given in the command line, as follows:

```
java SampleStock <HOST_URL> <SID> <SymbolList>
```

where:

Table 18–1 SampleStock Command Line Arguments

Argument	Description
HOST_URL	For example, egroup-dev3.us.oracle.com
SID	For example, db816
SymbolList	For example, 'ORCL INTC MSFT'

The output of `SampleStock` should be a table of data about requested stocks, such as the following:

```

|-----+-----|
|      | Time | 12:19PM |
|-----+-----|
|      | Price | 30 3/16 |
|-----+-----|
| ORCL | Change| +0.42%  |
|-----+-----|
|      | Volume| 34,272,000 |
|-----+-----|
|      | Time | 12:19PM |
|-----+-----|
|      | Price | 35 15/64 |
|-----+-----|
| INTC | Change| -2.80%  |
|-----+-----|
|      | Volume| 22,499,200 |
|-----+-----|
|      | Time | 12:19PM |
|-----+-----|
|      | Price | 63      |
|-----+-----|
| MSFT | Change| +0.10%  |
|-----+-----|
|      | Volume| 24,091,600 |
|-----+-----|

```

Dynamic Services Example 1: SampleStock (Java)

`SampleStock` takes the command line parameters and keeps them in the class data members. Then it calls `getQuotes()` to retrieve the quote data.

`SampleStock` abstracts the process of executing YahooPortofolio DS service by:

- Inputting a list of stock tickers and returning all relevant information in the format of `Java.util.Hashtable`.
- Input parameter, `symbolList` A string, consists of several stock symbols, which are separated by white space, such as, "ORCL INTC MSFT".
- The return `Hashtable` is indexed by symbol name. Each entry of `Hashtable` is also a `Hashtable` which is indexed by an information label, that is, "Time", "Price", "Change", or "Volume".
- `SampleStock` calls `printQuotes()` to display the results in the `Hashtable` in the format shown above.

```
import java.io.*;
import java.util.*;

// imports for handling XML docs
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;

// Dynamic Services imports
import oracle.ds.*;
import oracle.ds.comm.*;
import oracle.ds.registry.*;
import oracle.ds.comm.message.*;
import oracle.ds.driver.*;
import oracle.ds.utils.*;

/**
 * SampleStock code. It opens a Dynamic Service
 * connection, looks up a stock quote service,
 * executes it and stores the results in a hash table.
 */
public class SampleStock implements CommunicationMessageConstants
{
    // Usage message
    private static final String USAGE =
    (
        "Usage: java SampleStock <HOST_URL> <SID> <Symbol list>\n "+
        "where, \n"+
```

```

    "\tHOST_URL: i.e. egroup-dev3.us.oracle.com\n"+
    "\tSID: i.e. db816\n" +
    "\tSymbol list: i.e. 'ORCL MSFT SEBL'\n"
);

// The following are actually the parameters in the
// command line and some are hard coded.
private static String ms_szJdbcURL;
private static String ms_szUsername;
private static String ms_szPassword;
private static String ms_szServiceID;
private static String ms_szSymbolList;

// A utility to resolve the NamespacePrefix
// In our case, we hard coded the namespace of the prortfolio service
static private NSResolver ms_nsResolver = new NSResolver()
{
    public String resolveNamespacePrefix(String szPrefix)
    { return "http://www.portfolio.org/Portfolio/Response"; }
};

/**
 * The main function
 */
public static void main(String[] argv)
{
    // Strict: Take only 3 arguments
    if (argv.length != 3) { System.err.println(USAGE); return; }

    // Extract the command line arguments
    int iArgCounter = 0;
    ms_szJdbcURL = "jdbc:oracle:thin:@" + argv[iArgCounter++]
        + ":1521:" + argv[iArgCounter++];
    ms_szUsername = "dssys";
    ms_szPassword = "dssys";

    // Hardcoded service ID for "Yahoo StockQuote service"
    ms_szServiceID = "urn:com.yahoo:finance.portfolio03";
    ms_szSymbolList = argv[iArgCounter++];

    // Get quotes
    Hashtable ht = getQuotes(ms_szSymbolList);

    // Print quotes in a form
    printQuotes(ht);
}

```

```

} // end of main

/**
 * This function abstracts the process of executing the YahooPortofolio DS
 * service by taking in an input of a list of stock tickers and returning
 * all the relevant information in a java.util.Hashtable object.
 *
 * @param symbolList A white space delimited string containing the stock
 *                  symbols ( e.g. "ORCL INTIC MSFT" ).
 * @return A Hashtable indexed by the stock symbol, and each entry of the
 *         Hashtable is also a Hashtable which is indexed by the information
 *         label, i.e. "Time", "Price", "Change", "Volume".
 */
public static Hashtable getQuotes(String symbolList)
{
    Hashtable ht = new Hashtable();

    // If we are to use a session, a header field has to be set
    DSConnection dsconn = null;

    try
    {
        // First open the connection with the Direct Driver
        DSDriverManager.registerDriver("oracle.ds.driver.DSDirectDriver");
        dsconn = DSDriverManager.getConnection(ms_szJdbcURL);

        // Connect using your specified username/password
        dsconn.connect(ms_szUsername, ms_szPassword);
        System.err.println("==> Opened connection for "+ms_szUsername);

        // Lookup a service and obtaining the service request/response schemas
        DService dsServ = dsconn.lookupService(ms_szServiceID);

        // Make an XML request string from the symbol list
        String xmlRequest =
"<?xml version=\"1.0\"?>          \n" +
"<!-- Sample request of the Yahoo! portfolio service --> \n" +
"<PortfolioReq                    \n" +
"  xmlns=\"http://www.portfolio.org/Portfolio/Request\" >\n";

        StringTokenizer st = new StringTokenizer(ms_szSymbolList);
        while (st.hasMoreTokens()) {
            xmlRequest = xmlRequest + "<Symbol>" + st.nextToken() + "</Symbol>\n";
        }
    }
}

```

```
xmlRequest = xmlRequest + "</PortfolioReq>\n";

// Create a Request by requesting for a default request context
// from our Dynamic Services Connection
DSRequest dsReq = dsconn.createDSRequest(ms_szServiceID,
    new StringReader(xmlRequest));

// Execute synchronously, get the response and print it
DSResponse dsResp = null;
dsResp = dsconn.executeSynch(dsReq);

// Get the result XML
StringWriter sw = new StringWriter();
dsResp.writeResponse(sw);

// Instantiate a DOM Parser to parse the result
// to eventually get a Document
DOMParser xmlp = new DOMParser();
xmlp.parse( new StringReader(sw.toString()));
XMLDocument xmldoc = xmlp.getDocument();

// Get the list of "Quote" nodes
NodeList nlist = xmldoc.getElementsByTagName("Quote");
int nsym = nlist.getLength();

// For each 'Quote' node
for (int i = 0 ; i < nsym ; i ++ )
{
XMLNode qnode = (XMLNode) nlist.item(i);

// Make an entry from a quote node
Hashtable entry = new Hashtable();

entry.put("Time", qnode.valueOf("./P:Time", ms_nsResolver));
entry.put("Price", qnode.valueOf("./P:Price", ms_nsResolver));
entry.put("Change", qnode.valueOf("./P:Change", ms_nsResolver));
entry.put("Volume", qnode.valueOf("./P:Volume", ms_nsResolver));

// Insert into hash table
ht.put(qnode.valueOf("./P:Symbol", ms_nsResolver), entry);
} // end of for
}
catch (Exception e) { e.printStackTrace(); }

// Clean up job
```



```

finally
{
    // If I have a valid connection then do clean-up
    if(dsconn != null)
    {
// Now close the connectin
try { dsconn.close(); }
catch(Exception e) { e.printStackTrace(); }
    }
}

// Return the final result
return ht;
} // end of getQuotes

/**
 * This function outputs quotes in a Hashable to a form like
 * |-----+-----|
 * |      | Time | 12:19PM |
 * |      |-----+-----|
 * |      | Price | 30 3/16 |
 * | ORCL |-----+-----|
 * |      | Change| +0.42% |
 * |      |-----+-----|
 * |      | Volume| 34,272,000 |
 * |-----+-----|
 * |      | Time | 12:19PM |
 * |      |-----+-----|
 * |      | Price | 35 15/64 |
 * | INTC |-----+-----|
 * |      | Change| -2.80% |
 * |      |-----+-----|
 * |      | Volume| 22,499,200 |
 * |-----+-----|
 * |      | Time | 12:19PM |
 * |      |-----+-----|
 * |      | Price | 63 |
 * | MSFT |-----+-----|
 * |      | Change| +0.10% |
 * |      |-----+-----|
 * |      | Volume| 24,091,600 |
 * |-----+-----|
 *
 * @param ht A Java.util.Hastable that is the result of method getQuotes().
 */

```

```
public static void printQuotes(Hashtable ht)
{
    System.out.println("|-----+-----|");

    Enumeration e = ht.keys();

    // For each key in the Hashtable
    while (e.hasMoreElements())
    {
        // Key is a symbol
        String symbol = (String) e.nextElement();

        // Get an entry
        Hashtable entry = (Hashtable) ht.get(symbol);

        // -- Time
        System.out.println(" | Time | " +
            (String) entry.get("Time") + "\t\t| " );
        System.out.println(" |-----+-----|");

        // -- Price
        System.out.println(" | Price | " +
            (String) entry.get("Price") + " \t| " );
        System.out.println(" |" + symbol + " |-----+-----|");

        // -- Change
        System.out.println(" | Change| " +
            (String) entry.get("Change") + "\t\t| " );
        System.out.println(" |-----+-----|");

        // -- Volume
        System.out.println(" | Volume| " +
            (String) entry.get("Volume") + "\t| " );
        System.out.println(" |-----+-----|");
    } // end of while
} // end of printQuotes
}
```

Here is some typical XML code for SampleStock.java:

```
<PortfolioReq xmlns="http://www.portfolio.org/Portfolio/Request">
  <Symbol>ORCL</Symbol>
  <Symbol>INTC</Symbol>
</Portfolio>
```

Frequently Asked Questions (FAQs): Dynamic Services

How to Set Up a Language of Queuing and Sequencing Commands?

I am investigating setting up a small language for queueing and sequencing commands. XML has obvious benefits, so I would like to know if anybody is familiar with an existing dtd for that use? This is obviously not difficult, but if a standard DTD already exists, why reinvent the wheel? Desired commands include:

- fetch from queue
- perform xslt transform
- perform xsql process
- insert into queue
- send an email
- log a message
- set configuration parameter

Others may be required in the future.

Answer

Dynamic Services allows you to model all the commands you mentioned as XML based services.

You can define these services as well as the flow in which you can execute them. Dynamic Services engine will follow the flow (including sending email because it has SMTP services available inside the engine) and send you a completed response back. The engine is directly callable via Java/PLSQL APIs. You can also use the 'Dynamic Services Gateway' or equivalent, to accept requests over HTTP using SOAP or other SOAP like xml-rpc calls. Almost all parts of the engine are extensible to allow you to plug-in any transformation, protocol or execution flow according to your needs.

Other FAQs?

You can locate other Frequently Asked Questions (FAQs) with your DS software package at:

```
$ORACLE_HOME/ds/doc/dsfaq.txt
```

Oracle Syndication Server (OSS) and XML

Oracle Syndication Server (OSS) has a content syndication solution that generates one simple catalog for various content resources and aggregates any Internet data. It automatically pushes content updates to anywhere.

This chapter describes the following sections:

- [Introducing Oracle Syndication Services \(OSS\)](#)
- [OSS Features: e-Business Content Aggregation, Exchange, and Syndication](#)
- [Information and Content Exchange \(ICE\) Protocol](#)
- [OSS Architecture](#)
- [Interacting with Content Providers](#)
- [Interacting With Content Subscribers](#)

Introducing Oracle Syndication Services (OSS)

Content syndication can be applied to many application scenarios, including catalog exchange solutions in a B2B supply chain, and content providers syndicating to multiple portals. Oracle Syndication Server (OSS) has an extensible and scalable content syndication solution that offers the following support:

- Generates one simple catalog for various content resources
- Aggregates any Internet data
- Automatically pushes content updates to anywhere

Hence, OSS allows content syndication in B2B, B2C, and B2E, facilitating content exchange among providers, content delivery from providers to consumers, and content sharing in organizations.

See Also:

- <http://otn.oracle.com/products>
- [Chapter 18, "Using OracleAS Dynamic Services and XML"](#)

OSS Features: e-Business Content Aggregation, Exchange, and Syndication

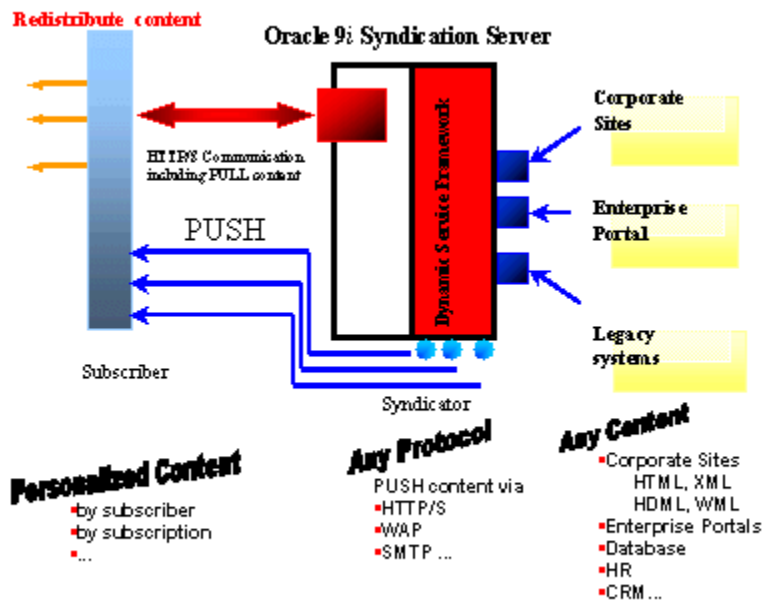
OSS provides content aggregation, exchange, and syndication for e-Businesses. OSS ensures your data is available, anywhere, and anytime. Its primary features are:

- **Aggregate content from any source:** OSS uses OracleAS Dynamic Services to extract or adapt information from any source, including existing Web sites, databases, enterprise applications, e-mail repositories, and legacy systems, to syndicate to its subscribers. These information and content sources are consolidated into one point of access known as a "content catalog", for any subscriber. See [Chapter 18, "Using OracleAS Dynamic Services and XML"](#) for information about OracleAS Dynamic Services.
- **Personalize subscriptions and content delivery for each subscriber:** OSS allows syndicators to personalize content for delivery based on profiles of subscribers, and to deliver content updates based on the subscription delivery policies.
- **Automate content delivery:** OSS pushes content to subscribers when information relevant to them changes. It allows a syndicator to schedule a "push" delivery for a subscriber. OSS uses Oracle9i Dynamic Services framework to push content over multiple channels.

- Transform content to adapt to a subscriber's format:** Because of the OracleAS Dynamic Services framework, OSS can transform content from any formatted source to a markup language suitable for each subscriber.

Figure 19–1 summarizes OSS features. Here you see OSS used to transport and redistribute any content, using any protocol, between syndicators and subscribers. It also offers content personalization. OSS is shown to facilitate data Push and Pull strategy. OSS includes administrative tools to manage subscriber profiles, content resources profiles, established subscriptions, and system monitoring.

Figure 19–1 Oracle Syndication Server (OSS) Features



Content Syndication

There are two roles in content syndication:

- Content Subscriber:** One of the two parties, that obtain information and content from a syndicator. This can be human subscribers, applications in organizations needing content feeds, or entire organizations that re-purpose the content they subscribe to.

- **Content Syndicator:** One of the two parties, that send information and content to a subscriber, retrieving the content from a third party content provider, or providing content in-house.

Oracle Syndication Server (OSS) can be configured to act as the content syndicator, content subscriber, or both.

Information and Content Exchange (ICE) Protocol

Information and Content Exchange (ICE) protocol is an XML-based specification to manage and automate the establishment of syndication relationships, data transfer, and results analysis in a content syndication scenario.

When combined with an industry specific vocabulary, ICE provides a complete solution for syndicating any type of information between information providers and subscribers. As a result, ICE can facilitate the controlled exchange and management of electronic assets between networked partners and affiliates. Applications based on ICE allow companies to easily construct syndicated publishing networks, Web superstores, and online reseller channels by establishing information and content exchange networks. ICE implementation features:

- **Message Definition:** ICE messages are XML based, and ICE 1.1 specification uses XML DTD to define the format of these messages, hence the protocol grammar is thereby established through this message definition.
- **Content Transport:** ICE protocol has been designed based XML document exchange. Each protocol message consists of a valid XML document, and the protocol involves sending such documents back and forth between syndicator and subscriber. The specification does not restrict ICE to any transport mechanism.
- **Security:** ICE implementations can achieve security using methods, such as encryption, at the transport level. Or applications can agree to send digitally signed content as items in ICE protocol. Or syndicators and subscribers can use certificates to authenticate each other.

ICE Operation Types

ICE protocol handles four types of operations:

- Subscription establishment and management. In ICE, a relationship between a Syndicator and a Subscriber begins with some form of subscription establishment. In ICE, the subscriber typically begins by obtaining a catalog of possible subscriptions (really, subscription offers) from the Syndicator. The

structure of a Catalog defined by ICE protocol consists of subscription offer groups. Each offer group has a set of offers as the finest unit for subscribers to choose from. For every offer, ICE protocol supports and defines the structure of associated delivery policies, usage reporting, presentation constraints, and business terms.

- Data delivery. The Subscriber then subscribes to particular subscriptions, possibly engaging in protocol parameter negotiation to arrive at mutually agreeable delivery methods and schedules.
- Event logs. The relationship then moves on to the steady state, where the primary message exchanges center on data delivery. ICE uses a package concept as a container mechanism for generic data items. ICE defines a sequenced package model allowing syndicators to support both incremental and full update models. ICE also defines push and pull data transfer models. Managing exceptional conditions and being able to diagnose problems is an important part of syndication management; accordingly, ICE defines a mechanism by which event logs can be automatically exchanged between (consenting) Subscribers and Syndicators.

Other Miscellaneous ICE Operations

Finally, ICE provides a number of mechanisms for supporting miscellaneous operations, such as the ability to renegotiate protocol parameters in an established relationship, the ability to send unsolicited ad-hoc notifications (i.e., textual messages) between systems (presumably ultimately targeted at administrators), the ability to query and ascertain the state of the relationship, etc.

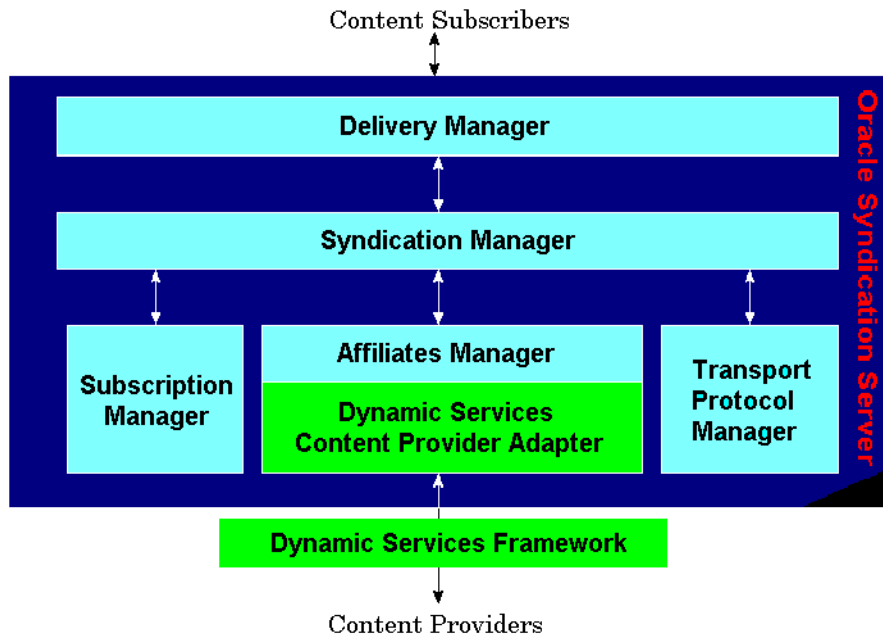
OSS Architecture

OSS is built on OracleAS Dynamic Services to adapt any content/information from Web sites, Internet Applications, or database, into an XML representation. It also transforms the XML into a markup language specific to the subscriber by means of XSL stylesheets.

Each content source is modeled as a set of services in OracleAS Dynamic Services. Information about the subscribers, content resources, and existing subscriptions is stored in OSS's registry. At runtime, OSS's engine processes subscriber content requests by invoking the corresponding components, and sends back content responses. On notification of content updates, OSS's engine checks its local registry to locate relevant subscribers and then pushes the updated information according to the delivery policy specified inside the subscription for each subscriber. Events from OSS are logged in an Oracle database.

Figure 19-2 illustrates Oracle Syndication Server's architecture overview and main functional components. Oracle Syndication Server (OSS) is built on OracleAS Dynamic Services framework using XML and Java. It communicates using a Java servlet, deployed in a servlet container with any industry standard Web listener, to receive and send messages. Information about subscribers and content resources as services are stored in OSS's registry on Oracle.

Figure 19-2 Oracle Syndication Server: Architecture Overview



OSS components are divided into two parts:

- Components that interface with content providers directly or through OracleAS Dynamic Services framework
- Syndication Server Engine that manages the interaction between content subscribers and the content providers.

Interacting with Content Providers

OSS interacts with content providers using OracleAS Dynamic Services(DS). OSS exposes content providers as a set of Dynamic Services services. The handling of a subscriber's request is mapped to a corresponding service execution inside the specified Dynamic Services Engine. OSS serves as a digital asset hub, aggregating content from all types of providers, regardless of location, access protocol, or content format.

Dynamic Services Content Provider Adapter (DSCPA)

DSCPA provides the interface for OSS's engine to access each registered content provider as a set of DS services. Enforced by the OSS, every published content provider service must comply with pre-defined DS interface. At runtime, OSS invokes the specified DSCPA to construct a DS service request from the subscriber's request. DSCPA then sends the service request to the specified DS engine for execution. The DS service engine's response is collected by DSCPA, marking the end of the transaction with a content provider. OSS requires each content provider to have the following minimum set of DS services:

- "Catalog" DS service. Used for content providers to provide catalog information of their available subscription offers.
- Subscription Approval DS service. Used as a way for certain content providers to approve subscriptions before they are stored in the OSSer. The service takes in a subscription request, forwarded by OSS, as input, and returns an approved subscription if the content provider agrees on all terms.
- Content Access DS service. Subscribers can initiate pulling content from OSS as well as having the content pushed to them.
- Subscription Cancellation DS service. Subscribers can cancel their subscriptions. When this occurs, OSS forwards the intent to an underlying "unsubscribe" DS service implemented by the content provider to allow the content provider to perform subscription clean ups.

Interacting With Content Subscribers

As OSS is implemented with ICE as the underlying mechanism of communication, a Content Subscriber Development Kit (CSDK) is bundled with OSS to facilitate subscribers in implementing their applications to communicate with any content syndicator using ICE. CSDK contains a client library that abstracts to some extent the formation of ICE messages from the subscriber's perspective, so applications

can be easily developed without being coupled with the underlying communication protocol with the server. Along with this client library is a set of API documentation that will further facilitate the process of content application development.

Delivering content to subscribers

Transport Protocol Manager (TPM) handles PULL and PUSH content deliveries from OSS to subscribers. In PUSH, subscribers can speak a protocol-specific language – most of them are XML-based markup languages.

To deliver data to a specific subscriber, TPM transforms the XML-formatted content to an appropriate subscriber-specific markup language and transports the content over a specified transport layer. This is done by a DS service model, one DS delivery service for each protocol. At runtime, the TPM selects a DS delivery service corresponding to a specified protocol, and executes it through the underlying DS framework.

Part VI

XDK for Java

Part VI describes how to access and use the XML components in XML Developer's Kit (XDK) for Java. Part VI contains the following chapters:

- [Chapter 20, "Using XML Parser for Java"](#)
- [Chapter 21, "Using XML Schema Processor for Java"](#)
- [Chapter 22, "XML Class Generator for Java"](#)

Note:

- XML-SQL Utility (XSU) is also considered part of the XDK for Java (and the XDK for PL/SQL). In this manual, XSU is described in Part II, [Chapter 7, "XML SQL Utility \(XSU\)"](#).
 - XSQL Servlet is considered part of XDK for Java. In this manual XSQL Servlet is described in Part IV, [Chapter 10, "XSQL Pages Publishing Framework"](#)
-
-

FAQs are included at the end of the following chapters:

- [Chapter 7: "Frequently Asked Questions \(FAQs\): XML SQL Utility \(XSU\)"](#)
- [Chapter 10: "Frequently Asked Questions \(FAQs\) - XSQL Servlet"](#)
- [Chapter 20: "Frequently Asked Questions \(FAQs\): XML Parser for Java"](#)
- [Chapter 22: "Frequently Asked Questions \(FAQs\): Class Generator for Java"](#)

Using XML Parser for Java

This chapter contains the following sections:

- [XML Parser for Java: Features](#)
- [Parsers Access XML Document's Content and Structure](#)
- [DOM and SAX APIs](#)
- [Running the XML Parser for Java Samples](#)
- [Using XML Parser for Java: DOMParser\(\) Class](#)
- [Using XML Parser for Java: DOMNamespace\(\) Class](#)
- [Using XML Parser for Java: SAXParser\(\) Class](#)
- [Using XML Parser for Java: XSLT Processor](#)
- [Using XML Parser for Java: SAXNamespace\(\) Class](#)
- [XML Parser for Java: Command Line Interface](#)
- [XML Extension Functions for XSLT Processing](#)
- [Frequently Asked Questions \(FAQs\): XML Parser for Java](#)

XML Parser for Java: Features

Oracle provides a set of XML parsers for Java, C, C++, and PL/SQL. Each of these parsers is a stand-alone XML component that parses an XML document (or a standalone DTD or XML Schema) so that it can be processed by an application. Library and command-line versions are provided supporting the following standards and features:

- **XML.** W3C XML 1.0 Recommendation
- **DOM.** Integrated DOM (Document Object Model) API, compliant with:
 - W3C DOM 1.0 Recommendation
 - W3C DOM 2.0 CORE Recommendation
 - W3C DOM 2.0 Traversal Recommendation, including Treewalker, Node Iterator, and Node Filter.

These APIs permit applications to access and manipulate an XML document as a tree structure in memory. This interface is used by such applications as editors.

- **SAX.** Integrated SAX (Simple API for XML) API, compliant with the SAX 2.0 recommendation. These APIs permit an application to process XML documents using an event-driven model.
- W3C Proposed Recommendation for XML Namespaces 1.0 thereby avoiding name collision, increasing reusability and easing application integration. Supports Oracle XML Schema Processor. See also <http://www.w3.org/TR/1999/REC-xml-names-19990114/>
- **XSLT.** XSLT Processor for Java includes the following features:
 - Integrated support for W3C XSLT 1.1 Working Draft
 - Provides new APIs to get XSL Transformation as SAX Output
- XML Schema Processor. See [Chapter 21, "Using XML Schema Processor for Java"](#). Supports XML Schema Processor that parses and validates XML files against an XML Schema Definition file (.xsd). It includes the following features:
 - Built on the XML Parser for Java v2
 - Supports the three parts of the XML Schema Working Draft
 - * Part 0: Primer XML Schema
 - * Part 1: Structures XML Schema

* Part 2: Datatypes

- Runs on Oracle and Oracle Application Server

Additional features include:

- Validating and non-validating modes
- Built-in error recovery until fatal error
- DOM extension APIs for document creation

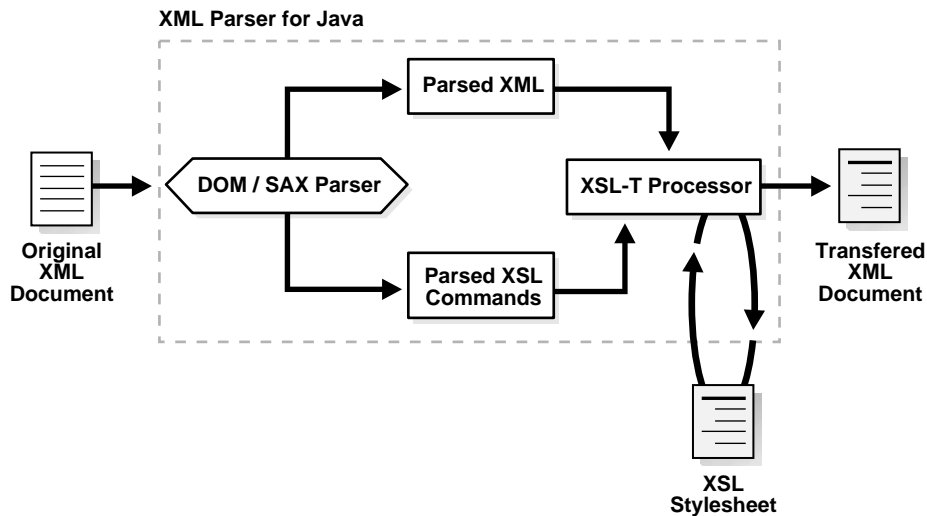
The parsers are available on all Oracle platforms.

[Figure 20–1](#) shows an XML document inputting XML Parser for Java. The DOM or SAX parser interface parses the XML document. The parsed XML is then transferred to the application for further processing.

If a stylesheet is used, the DOM or SAX interface also parses and outputs the XSL commands. These are sent together with the parsed XML to the XSLT Processor where the selected stylesheet is applied and the transformed (new) XML document is then output.

See Also: [Appendix C, "XDK for Java: Specifications and Cheat Sheets"](#).

Figure 20–1 Oracle XML Parser



DOM and SAX APIs are explained in ["DOM and SAX APIs"](#).

The classes and methods used to parse an XML document are illustrated in the following diagrams:

- [Figure 20–4, "XML Parser for Java: DOMParser\(\)"](#)
- [Figure 20–5, "Using SAXParser\(\) Class"](#)

The classes and methods used by the XSLT Processor to apply stylesheets are illustrated in the following diagram:

- [Figure 20–6, "XSLProcessor Class Process"](#)

XSL Transformation (XSLT) Processor

The V2 versions of the XML Parsers include an integrated XSL Transformation (XSLT) Processor for transforming XML data using XSL stylesheets. Using the XSLT processor, you can transform XML documents from XML to XML, XML to HTML, or to virtually any other text-based format. See [Figure 20–1](#).

The processor supports the following standards and features:

- Compliant with the W3C XSL Transform Proposed Recommendation 1.0
- Compliant with the W3C XPath Proposed Recommendation 1.0

- Integrated into the XML Parser for improved performance and scalability
- Available with library and command-line interfaces for Java, C, C++, and PL/SQL

Namespace Support

The Java, C, and C++ XML parsers also support XML Namespaces. Namespaces are a mechanism to resolve or avoid name collisions between element types (tags) or attributes in XML documents.

This mechanism provides "universal" namespace element types and attribute names whose scope extends beyond this manual.

Such tags are qualified by uniform resource identifiers (URIs), such as:

```
<oracle:EMP xmlns:oracle="http://www.oracle.com/xml" />
```

For example, namespaces can be used to identify an Oracle <EMP> data element as distinct from another company's definition of an <EMP> data element.

This enables an application to more easily identify elements and attributes it is designed to process. The Java, C, and C++ parsers support namespaces by being able to recognize and parse universal element types and attribute names, as well as unqualified "local" element types and attribute names.

See Also: [Chapter 21, "Using XML Schema Processor for Java"](#)

Oracle XML Parsers Support Four Validation Modes

The Java, C, and C++ parsers can parse XML in validating or non-validating modes.

- **Non-Validating Mode.** The parser verifies that the XML is well-formed and parses the data into a tree of objects that can be manipulated by the DOM API.
- **DTD Validating Mode.** The parser verifies that the XML is well-formed and validates the XML data against the DTD (if any).
- **Partial Validation Mode.** Partial validation validates an input XML document as per the DTD if a DTD or XMLS Schema is present else it will be in NON Validation mode.
- **Schema Validation Mode.** The XML Document is validated as per the XML Schema specified for the document.
- **Auto Validation Mode.** In this mode the parser does its best to validate with whatever is available. If DTD is available, it is set to DTD_VALIDATION, if

Schema is present then it is set to SCHEMA_VALIDATION. If none is available, it is set to NON_VALIDATING mode.

Validation involves checking whether or not the attribute names and element tags are legal, whether nested elements belong where they are, and so on.

See Also: *Oracle9i XML Reference*

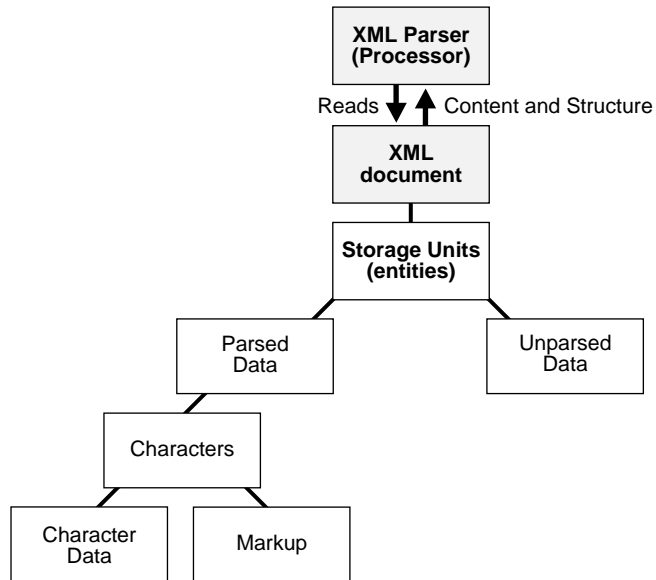
Parsers Access XML Document's Content and Structure

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 parsing process is illustrated in [Figure 20-2](#).

Figure 20-2 XML Parsing Process

DOM and SAX APIs

XML APIs generally fall into the following two categories:

- Event-based
- Tree-based

See [Figure 20–3](#). Consider the following simple XML document:

```
<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
      <ENAME>MARY</ENAME>
    </EMP>
    <EMP>
      <ENAME>SCOTT</ENAME>
    </EMP>
  </EMPLIST>
```

DOM: Tree-Based API

A tree-based API (such as Document Object Model, DOM) builds an in-memory tree representation of the XML document. It provides classes and methods for an application to navigate and process the tree.

In general, the DOM interface is most useful for structural manipulations of the XML tree, such as reordering elements, adding or deleting elements and attributes, renaming elements, and so on. For example, for the XML document above, the DOM creates an in-memory tree structure as shown in [Figure 20–3](#).

SAX: Event -Based API

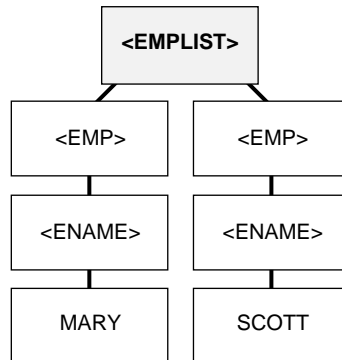
An event-based API (such as SAX) uses calls to report parsing events to the application. The application deals with these events through customized event handlers. Events include the start and end of elements and characters.

Unlike tree-based APIs, event-based APIs usually do not build in-memory tree representations of the XML documents. Therefore, in general, SAX is useful for applications that do not need to manipulate the XML tree, such as search operations, among others.

The above XML document becomes a series of linear events as shown in [Figure 20–3](#).

Figure 20–3 Comparing DOM (Tree-Based) and SAX (Event-Based) APIs**XML Document**

```
<?XML Version = "1.0"?>
<EMPLIST>
  <EMP>
    <ENAME>MARY</ENAME>
  </EMP>
  <EMP>
    <ENAME>SCOTT</ENAME>
  </EMP>
</EMPLIST>
```

The DOM interface creates a TREE structure based on the XML Document

Useful for applications that include changes eg. reordering, adding, or deleting elements.

The SAX interface creates a series of linear events based on the XML document

start document

start element: EMPLIST
 start element: EMP
 start element: ENAME
 characters: MARY
 end element: EMP

start element: EMP
 start element: ENAME
 characters: SCOTT
 end element: EMP

end element: EMPLIST
 end document

Useful for applications such as search and retrieval that do not change the "XML tree".

Guidelines for Using DOM and SAX APIs

Here are some guidelines for using the DOM and SAX APIs:

DOM:

- Use the DOM API when you need to use random access.
- DOM consumes more memory.
- Use DOM when you are performing transformations.
- Use DOM when you want to have tree iterations and need to walk through the entire document tree.
- When using the DOM interface, try to use more attributes over elements in your XML, to reduce the pipe size.

SAX:

Use the SAX API when your data is mostly streaming data.

XML Parser and Data Compression

Oracle XML Parser can also compress XML documents. Using the compression feature, an in-memory DOM tree or the SAX events generated from an XML document can be compressed to generate a binary compressed output.

The compressed stream generated from DOM and SAX are compatible, that is, the compressed stream generated from SAX could be used to generate the DOM tree and vice versa. The compression is based on tokenizing the XML tags. This is based on the assumption that XML files typically have repeated tags and tokenizing the tags compresses the data. The compression depends on the type of input XML document — the larger the number of tags, the less the text content, and the better the compression.

As with XML documents in general, you can store the *compressed* XML data output as a CLOB (Character Large Object) in the database.

XML Serialization/Compression

An XML document is compressed into a binary stream by means of the serialization of an in-memory DOM tree. When a large XML document is parsed and a DOM tree is created in memory corresponding to it, it may be difficult to satisfy memory requirements and this could affect performance. The XML document is compressed into a byte stream and stored in an in-memory DOM tree. This can be expanded at a later time into a DOM tree without performing validation on the XML data stored in the compressed stream.

The compressed stream can be treated as a serialized stream, but note that the information in the stream is more controlled and managed, compared to the compression implemented by Java's default serialization.

In this release, there are two kinds of XML compressed streams:

- **SAX based Compression:** The compressed stream is generated when an XML file is parsed using a SAX Parser. SAX events generated by the SAX Parser are handled by the SAX Compression utility. It handles the SAX events to generate a compressed binary stream. When the binary stream is read back, the SAX events are generated.
- **DOM based compression:** The in-memory DOM tree, corresponding to a parsed XML document, is serialized, and a compressed XML output stream is generated. This serialized stream when read back regenerates the DOM tree.

The compressed stream is generated using SAX events and that generated using DOM serialization are compatible. You can use the compressed stream generated

by SAX events to create a DOM tree and vice versa. The compression algorithm used is based on tokenizing the XML tag's. The assumption is that any XML file has repeated number of tags and therefore tokenizing the tags will give considerable compression.

See Also:

- [Chapter 2, "Modeling and Design Issues for Oracle XML Applications", "Loading XML into a Database"](#) on page 2-13.
- [Chapter 5, "Database Support for XML"](#)

Upgrading XDK for Java

Upgrading XDK for Java from a Previous Release to Oracle

If you already have XDK for Java installed, and are upgrading to Oracle, follow these steps:

1. Make sure you have successfully upgraded JServer.
2. Change to the `ORACLE_HOME/rdbms/admin` directory.
3. Start SQL*Plus.
4. Connect to the database instance as a user with SYSDBA privileges.
5. Run STARTUP:

```
SQL> STARTUP
```

You may need to use the `PFILE` option to specify the location of your initialization parameter file.

6. Run the appropriate upgrade script depending on the release from which you are upgrading.

If you are upgrading from release 8.1.5, run `xmlu815.sql`:

```
SQL> @xmlu815.sql
```

If you are upgrading from release 8.1.6, run `xmlu816.sql`:

```
SQL> @xmlu816.sql
```

If you are upgrading from release 8.1.7, run `xmlu817.sql`:

```
SQL> @xmlu817.sql
```

7. Shut down all instances using SHUTDOWN:

```
SQL> SHUTDOWN
```

8. Exit SQL*Plus.

The XDK for Java component is upgraded to the new release.

Upgrading Session Namespace, CORBA, and OSE

1. Make sure you have successfully upgraded JServer and XDK for Java.
2. At a system prompt, change to the `ORACLE_HOME/javavm/install` directory.

Upgrading JSP

If the Oracle system has JSP installed, then complete the following steps:

1. Make sure you have successfully upgraded JServer, XDK for Java, and Session Namespace, CORBA, and OSE.
2. At a system prompt, change to the `ORACLE_HOME/javavm/install` directory.

Downgrading to Oracle Release 8.1

See Chapter 13 of the *Oracle9i Migration* manual.

Running the XML Parser for Java Samples

[Table 20–1](#) lists the XML Parser for Java examples provided with XDK for Java software. The samples are located in the `sample/` subdirectory. They illustrate how to use Oracle XML Parser for Java.

Table 20–1 XML Parser for Java Samples

Name of Sample File	Description
DOMSample.java	A sample application using DOM APIs.
SAXSample.java	A sample application using SAX APIs.
XSLSample.java	A sample application using XSL APIs.

Table 20–1 XML Parser for Java Samples (Cont.)

Name of Sample File	Description
DOMNamespace.java	A sample application using Namespace extensions to DOM APIs.
SAXNamespace.java	A sample application using Namespace extensions to SAX APIs.

Note that because some package names are different in V2, different files were generated to show the differences between V2 and V1 of the XML Parser for Java.

To run the sample programs:

1. Use “**make**” to generate .class files.
2. Add `xmlparserv2.jar` and the current directory to the CLASSPATH.
3. Run the sample program for DOM/SAX APIs as follows:

```
java <classname> <sample xml file>
```

4. Run the sample program for XSL APIs as follows:

```
java XSLSample <sample xsl file> <sample xml file>
```

A few XML files such as `class.xml`, `empl.xml`, and `family.xml`, are provided as test cases.

XSL stylesheet `iden.xsl`, can be used to achieve an identity transformation of the supplied XML files:

- `class.xml`
- `NSExample.xml`
- `family.xml`
- `empl.xml`

XML Parser for Java - XML Sample 1: class.xml

```
<?xml version = "1.0"?>
<!DOCTYPE course [
<!ELEMENT course (Name, Dept, Instructor, Student)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Instructor (Name)>
```

```
<!ELEMENT Student (Name*)>
]>
<course>
<Name>Calculus</Name>
<Dept>Math</Dept>
<Instructor>
<Name>Jim Green</Name>
</Instructor>
<Student>
<Name>Jack</Name>
<Name>Mary</Name>
<Name>Paul</Name>
</Student>
</course>
```

XML Parser for Java - XML Example 2: Using DTD employee — employee.xml

```
<?xml version="1.0"?>
<!DOCTYPE employee [
<!ELEMENT employee (Name, Dept, Title)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
]>
<employee>
<Name>John Goodman</Name>
<Dept>Manufacturing</Dept>
<Title>Supervisor</Title>
</employee>
```

XML Parser for Java - XML Example 3: Using DTD family.dtd — family.xml

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE family SYSTEM "family.dtd">
<family lastname="Smith">
<member memberid="m1">Sarah</member>
<member memberid="m2">Bob</member>
<member memberid="m3" mom="m1" dad="m2">Joanne</member>
<member memberid="m4" mom="m1" dad="m2">Jim</member>
</family>
```

DTD: family.dtd

```

<!ELEMENT family (member*)>
<!ATTLIST family lastname CDATA #REQUIRED>
<!ELEMENT member (#PCDATA)>
<!ATTLIST member memberid ID #REQUIRED>
<!ATTLIST member dad IDREF #IMPLIED>
<!ATTLIST member mom IDREF #IMPLIED>

```

XML Parser for Java — XSL Example 1: XSL (iden.xsl)

```

<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="*|@*|comment()|processing-instruction()|text() ">
    <xsl:copy>
      <xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>

```

XML Parser for Java - DTD Example 1: (NSExample)

```

<!DOCTYPE doc [
<!ELEMENT doc (child*)>
<!ATTLIST doc xmlns:nsprefix CDATA #IMPLIED>
<!ATTLIST doc xmlns CDATA #IMPLIED>
<!ATTLIST doc nsprefix:al CDATA #IMPLIED>
<!ELEMENT child (#PCDATA)>
]>
<doc nsprefix:al = "v1" xmlns="http://www.w3c.org"
xmlns:nsprefix="http://www.oracle.com">
<child>
This element inherits the default Namespace of doc.
</child>
</doc>

```

Using XML Parser for Java: DOMParser() Class

To write DOM based parser applications you can use the following classes:

- `DOMNamespace()` class
- `DOMParser()` class
- `XMLParser()` class

Since `DOMParser` extends `XMLParser`, all methods of `XMLParser` are also available to `DOMParser`. [Figure 20–4](#) shows the main steps you need when coding with the `DOMParser()` class:

- ***Without DTD Input***

1. A new `DOMParser()` class is called. Available properties to use with this class are:
 - * `setValidateMode`
 - * `setPreserveWhiteSpace`
 - * `setDocType`
 - * `setBaseURL`
 - * `showWarnings`
2. The results of 1) are passed to `XMLParser.parse()` along with the XML input. The XML input can be a file, a string buffer, or URL.
3. Use the `XMLParser.getDocument()` method.
4. Optionally, you can apply other DOM methods such as:
 - * `print()`
 - * `DOMNamespace()` methods
5. The Parser outputs the DOM tree XML (parsed) document.
6. Optionally, use `DOMParser.reset()` to clean up any internal data structures, once the Parser has finished building the DOM tree.

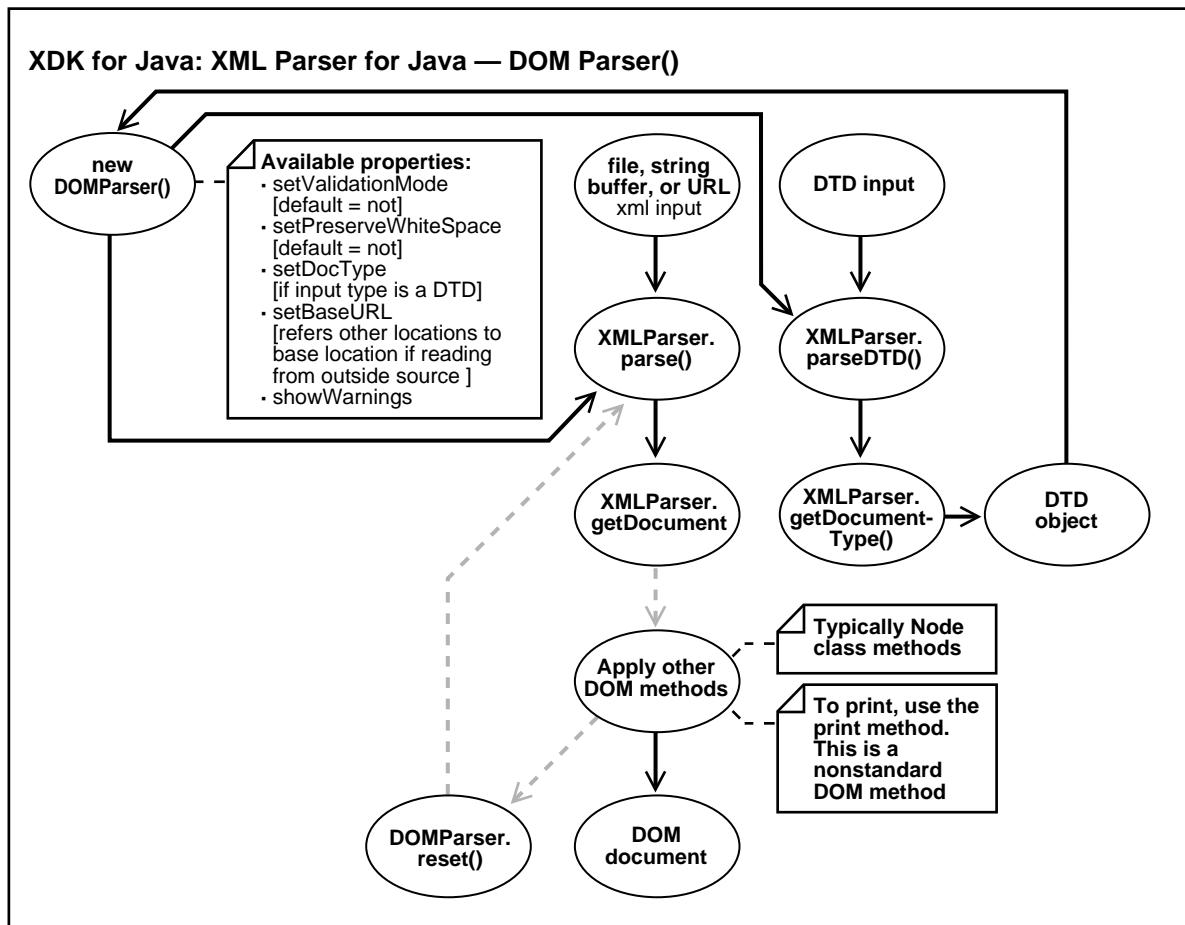
- ***With a DTD Input***

1. A new `DOMParser()` class is called. The available properties to apply to this class are:
 - * `setValidateMode`

- * `setPreserveWhiteSpace`
 - * `setDocType`
 - * `setBaseURL`
 - * `showWarnings`
2. The results of 1) are passed to `XMLParser.parseDTD()` method along with the DTD input.
 3. `XMLParser.getDocumentType()` method then sends the resulting DTD object back to the new `DOMParser()` and the process continues until the DTD has been applied.

The example, "[XML Parser for Java Example 1: Using the Parser and DOM API \(DomSample.java\)](#)", shows how to use `DOMParser()` class.

Figure 20–4 XML Parser for Java: DOMParser()



XML Parser for Java Example 1: Using the Parser and DOM API (DomSample.java)

The examples represent the way we write code so it is required to present the examples with Java coding standards (like all imports expanded), with documentation headers before the methods, and so on.

```
// This file demonstrates a simple use of the parser and DOM API.
// The XML file given to the application is parsed.
```



```
// The elements and attributes in the document are printed.
// This demonstrates setting the parser options.
//

import java.io.*;
import java.net.*;
import org.w3c.dom.*;
import org.w3c.dom.Node;

import oracle.xml.parser.v2.*;

public class DOMSample
{
    static public void main(String[] argv)
    {
        try
        {
            if (argv.length != 1)
            {
                // Must pass in the name of the XML file.
                System.err.println("Usage: java DOMSample filename");
                System.exit(1);
            }

            // Get an instance of the parser
            DOMParser parser = new DOMParser();

            // Generate a URL from the filename.
            URL url = createURL(argv[0]);

            // Set various parser options: validation on,
            // warnings shown, error stream set to stderr.
            parser.setErrorStream(System.err);
            parser.setValidationMode(DTD_validation);
            parser.showWarnings(true);

            // Parse the document.
            parser.parse(url);

            // Obtain the document.
            XMLDocument doc = parser.getDocument();

            // Print document elements
            System.out.print("The elements are: ");
            printElements(doc);
        }
    }
}
```

```
        // Print document element attributes
        System.out.println("The attributes of each element are: ");
        printElementAttributes(doc);
        parser.reset();
    }
    catch (Exception e)
    {
        System.out.println(e.toString());
    }
}

static void printElements(Document doc)
{
    NodeList nl = doc.getElementsByTagName("*");
    Node n;

    for (int i=0; i<nl.getLength(); i++)
    {
        n = nl.item(i);
        System.out.print(n.getNodeName() + " ");
    }

    System.out.println();
}

static void printElementAttributes(Document doc)
{
    NodeList nl = doc.getElementsByTagName("*");
    Element e;
    Node n;
    NamedNodeMap nnm;

    String attrname;
    String attrval;
    int i, len;

    len = nl.getLength();
    for (int j=0; j < len; j++)
    {
        e = (Element)nl.item(j);
        System.out.println(e.getTagName() + ":");
        nnm = e.getAttributes();
        if (nnm != null)
        {
```

```
        for (i=0; i<nm.getLength(); i++)
        {
            n = nm.item(i);
            attrname = n.getNodeName();
            attrval = n.getNodeValue();
            System.out.print(" " + attrname + " = " + attrval);
        }
    }
    System.out.println();
}

static URL createURL(String fileName)
{
    URL url = null;
    try
    {
        url = new URL(fileName);
    }
    catch (MalformedURLException ex)
    {
        File f = new File(fileName);
        try
        {
            String path = f.getAbsolutePath();
            String fs = System.getProperty("file.separator");
            if (fs.length() == 1)
            {
                char sep = fs.charAt(0);
                if (sep != '/')
                    path = path.replace(sep, '/');
                if (path.charAt(0) != '/')
                    path = '/' + path;
            }
            path = "file://" + path;
            url = new URL(path);
        }
        catch (MalformedURLException e)
        {
            System.out.println("Cannot create url for: " + fileName);
            System.exit(0);
        }
    }
    return url;
}
```

```
}
```

Comments on DOMParser() Example 1

See also [Figure 20-4](#). The following provides comments for Example 1:

1. Declare a new `DOMParser()`. In Example 1, see the line:

```
DOMParser parser = new DOMParser();
```

This class has several properties you can use. Here the example uses:

```
parser.setErrorStream(System.err);  
parser.setValidationMode(DTD_validation);  
parser.showWarnings(true);
```

2. The XML input is a URL as declared by:

```
URL url = createURL(argv[0])
```

3. The XML document is input as a URL. This is parsed using `parser.parse()`:

```
parser.parse(url);
```

4. Gets the document:

```
XMLDocument doc = parser.getDocument();
```

5. Applies other DOM methods. In this case:

- Node class methods:

- * `getElementsByTagName()`
- * `getAttributes()`
- * `getNodeName()`
- * `getNodeValue()`

- Method, `createURL()` to convert the string name into a URL.

6. `parser.reset()` is called to clean up any data structure created during the parse process, after the DOM tree has been created. Note that this is a new method with this release.
7. Generates the DOM tree (parsed XML) document for further processing by your application.

Note: No DTD input is shown in Example 1.

Using XML Parser for Java: DOMNamespace() Class

Figure 20-3 illustrates the main processes involved when parsing an XML document using the DOM interface. The DOMNamespace() method is applied in the parser process at the “bubble” that states “Apply other DOM methods”. The following example illustrates how to use DOMNamespace():

- ["XML Parser for Java Example 2: Parsing a URL — DOMNamespace.java"](#)

XML Parser for Java Example 2: Parsing a URL — DOMNamespace.java

```
// This file demonstrates a simple use of the parser and Namespace
// extensions to the DOM APIs.
// The XML file given to the application is parsed and the
// elements and attributes in the document are printed.
//

import java.io.*;
import java.net.*;

import oracle.xml.parser.v2.DOMParser;

import org.w3c.dom.*;
import org.w3c.dom.Node;

// Extensions to DOM Interfaces for Namespace support.
import oracle.xml.parser.v2.XMLElement;
import oracle.xml.parser.v2.XMLAttr;

public class DOMNamespace
{
    static public void main(String[] argv)
    {
        try
        {
            if (argv.length != 1)
            {
                // Must pass in the name of the XML file.
                System.err.println("Usage: DOMNamespace filename");
                System.exit(1);
            }
        }
    }
}
```

```
    }

    // Get an instance of the parser
    Class cls = Class.forName("oracle.xml.parser.v2.DOMParser");
    DOMParser parser = (DOMParser)cls.newInstance();

// Generate a URL from the filename.
URL url = createURL(argv[0]);

// Parse the document.
    parser.parse(url);

    // Obtain the document.
    Document doc = parser.getDocument();

    // Print document elements
    printElements(doc);

    // Print document element attributes
    System.out.println("The attributes of each element are: ");
    printElementAttributes(doc);
}
catch (Exception e)
{
    System.out.println(e.toString());
}
}

static void printElements(Document doc)
{
    NodeList nl = doc.getElementsByTagName("*");
    XMLElement nsElement;

    String qName;
    String localName;
    String nsName;
    String expName;

    System.out.println("The elements are: ");
    for (int i=0; i < nl.getLength(); i++)
    {
        nsElement = (XMLElement)nl.item(i);

        // Use the methods getQualifiedName(), getLocalName(), getNamespace()
        // and getExpandedName() in NSName interface to get Namespace
```

```
// information.

qName = nsElement.getQualifiedName();
System.out.println(" ELEMENT Qualified Name:" + qName);

localName = nsElement.getLocalName();
System.out.println(" ELEMENT Local Name      :" + localName);

nsName = nsElement.getNamespace();
System.out.println(" ELEMENT Namespace       :" + nsName);

expName = nsElement.getExpandedName();
System.out.println(" ELEMENT Expanded Name  :" + expName);
}

System.out.println();
}

static void printElementAttributes(Document doc)
{
    NodeList nl = doc.getElementsByTagName("*");
    Element e;
    XMLAttr nsAttr;
    String attrname;
    String attrval;
    String attrqname;

    NamedNodeMap nnm;
    int i, len;
    len = nl.getLength();
    for (int j=0; j < len; j++)
    {
        e = (Element) nl.item(j);
        System.out.println(e.getTagName() + ":");
        nnm = e.getAttributes();

        if (nnm != null)
        {
            for (i=0; i < nnm.getLength(); i++)
            {
                nsAttr = (XMLAttr) nnm.item(i);

                // Use the methods getQualifiedName(), getLocalName(),
                // getNamespace() and getExpandedName() in NSName
                // interface to get Namespace information.
            }
        }
    }
}
```

```
        attrname = nsAttr.getExpandedName();
        attrqname = nsAttr.getQualifiedName();
        attrval = nsAttr.getNodeValue();

        System.out.println(" " + attrqname + "(" + attrname + ")" + " = "
+ attrval);
    }
}
System.out.println();
}
}

static URL createURL(String fileName)
{
    URL url = null;
    try
    {
        url = new URL(fileName);
    }
    catch (MalformedURLException ex)
    {
        File f = new File(fileName);
        try
        {
            String path = f.getAbsolutePath();
            String fs = System.getProperty("file.separator");
            if (fs.length() == 1)
            {
                char sep = fs.charAt(0);
                if (sep != '/')
                    path = path.replace(sep, '/');
                if (path.charAt(0) != '/')
                    path = '/' + path;
            }
            path = "file://" + path;
            url = new URL(path);
        }
        catch (MalformedURLException e)
        {
            System.out.println("Cannot create url for: " + fileName);
            System.exit(0);
        }
    }
    return url;
}
```



```

    }
}

```

Note: No DTD is input is shown in Example 2.

Using XML Parser for Java: SAXParser() Class

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 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.
- It adds Namespace support, which is required for many higher-level XML standards.

Table 20-2 lists the class SAXParser() methods.

Table 20-2 Class SAXParser() Methods

Method	Description
getContentHandler()	Returns the current content handler.
getDTDHandler()	Returns the current DTD handler.
getEntityResolver()	Returns the current entity resolver.
getErrorHandler()	Returns the current error handler.
getFeature(java.lang.String name)	Looks up the value of a feature.
getProperty(java.lang.String name)	Looks up the value of a property.
setContentHandler(ContentHandler handler)	Allows an application to register a content event handler.

Table 20–2 Class SAXParser() Methods(Cont.)

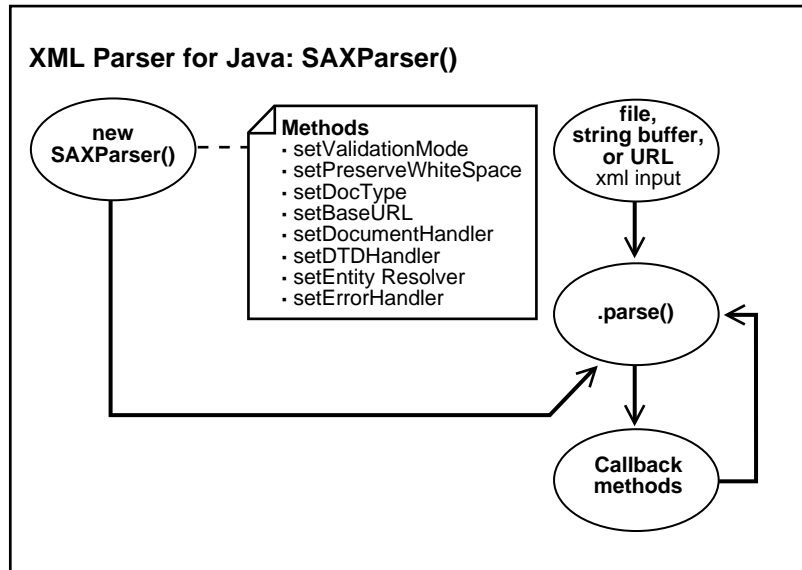
Method	Description
setDocumentHandler(DocumentHandler handler)	Deprecated. as of SAX2.0 - Replaced by setContentHandler
setDTDHandler(DTDHandler handler)	Allows an application to register a DTD event handler.
setEntityResolver(EntityResolver resolver)	Allows an application to register an entity resolver.
setErrorHandler(ErrorHandler handler)	Allows an application to register an error event handler.
setFeature(java.lang.String name, boolean value)	Sets the state of a feature.
setProperty(java.lang.String name, java.lang.Object value)	Sets the value of a property.

Figure 20–5 shows the main steps for coding with the SAXParser() class.

1. Declare a new `DOMParser()` class. Table 20–2 lists the available methods.
2. The results of 1) are passed to `.parse()` along with the XML input in the form of a file, string, or URL.
3. Parse methods return when parsing completes. Meanwhile the process waits for an event-handler callback to return before reporting the next event.
4. The parsed XML document is available for further processing by your application.

The example, "[XML Parser for Java Example 3: Using the Parser and SAX API \(SAXSample.java\)](#)", illustrates how you can use SAXParser() class and several handler interfaces.

Figure 20–5 Using SAXParser() Class



XML Parser for Java Example 3: Using the Parser and SAX API (SAXSample.java)

```
// This file demonstrates a simple use of the parser and SAX API.
// The XML file given to the application is parsed and
// prints out some information about the contents of this file.
//
```

```
import org.xml.sax.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;

public class SAXSample extends HandlerBase
{
    // Store the locator
    Locator locator;

    static public void main(String[] argv)
    {
        try
        {
```

```
    if (argv.length != 1)
    {
        // Must pass in the name of the XML file.
        System.err.println("Usage: SAXSample filename");
        System.exit(1);
    }
    // (1) Create a new handler for the parser
    SAXSample sample = new SAXSample();

    // (2) Get an instance of the parser
    Parser parser = new SAXParser();

    // (3) Set Handlers in the parser
    parser.setDocumentHandler(sample);
    parser.setEntityResolver(sample);
    parser.setDTDHandler(sample);
    parser.setErrorHandler(sample);

    // (4) Convert file to URL and parse
    try
    {
        parser.parse(fileToURL(new File(argv[0])).toString());
    }
    catch (SAXParseException e)
    {
        System.out.println(e.getMessage());
    }
    catch (SAXException e)
    {
        System.out.println(e.getMessage());
    }
}
catch (Exception e)
{
    System.out.println(e.toString());
}
}

static URL fileToURL(File file)
{
    String path = file.getAbsolutePath();
    String fSep = System.getProperty("file.separator");
    if (fSep != null && fSep.length() == 1)
        path = path.replace(fSep.charAt(0), '/');
    if (path.length() > 0 && path.charAt(0) != '/')

```

```
        path = '/' + path;
    try
    {
        return new URL("file", null, path);
    }
    catch (java.net.MalformedURLException e)
    {
        throw new Error("unexpected MalformedURLException");
    }
}

/////////////////////////////////////////////////////////////////
// (5) Sample implementation of DocumentHandler interface.
/////////////////////////////////////////////////////////////////

public void setDocumentLocator (Locator locator)
{
    System.out.println("SetDocumentLocator:");
    this.locator = locator;
}

public void startDocument()
{
    System.out.println("StartDocument");
}

public void endDocument() throws SAXException
{
    System.out.println("EndDocument");
}

public void startElement(String name, AttributeList atts)
                                                throws SAXException
{
    System.out.println("StartElement:"+name);
    for (int i=0;i<atts.getLength();i++)
    {
        String aname = atts.getName(i);
        String type = atts.getType(i);
        String value = atts.getValue(i);

        System.out.println("    "+aname+"("+type+")"+"="+value);
    }
}

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

```

    {
        System.out.println("EndElement:"+name);
    }

    public void characters(char[] cbuf, int start, int len)
    {
        System.out.print("Characters:");
        System.out.println(new String(cbuf,start,len));
    }

    public void ignorableWhitespace(char[] cbuf, int start, int len)
    {
        System.out.println("IgnorableWhiteSpace");
    }

    public void processingInstruction(String target, String data)
        throws SAXException
    {
        System.out.println("ProcessingInstruction:"+target+" "+data);
    }

    ////////////////////////////////////////////////////////////////////
    // (6) Sample implementation of the EntityResolver interface.
    ////////////////////////////////////////////////////////////////////

    public InputSource resolveEntity (String publicId, String systemId)
        throws SAXException
    {
        System.out.println("ResolveEntity:"+publicId+" "+systemId);
        System.out.println("Locator:"+locator.getPublicId()+" "+
            locator.getSystemId()+
            " "+locator.getLineNumber()+" "+locator.getColumnNumber());
        return null;
    }

    ////////////////////////////////////////////////////////////////////
    // (7) Sample implementation of the DTDHandler interface.
    ////////////////////////////////////////////////////////////////////

    public void notationDecl (String name, String publicId, String systemId)
    {
        System.out.println("NotationDecl:"+name+" "+publicId+" "+systemId);
    }

```

```

public void unparsedEntityDecl (String name, String publicId,
                               String systemId, String notationName)
{
    System.out.println("UnparsedEntityDecl:"+name + " "+publicId+" "+
                      systemId+" "+notationName);
}

////////////////////////////////////
// (8) Sample implementation of the ErrorHandler interface.
////////////////////////////////////

public void warning (SAXParseException e)
    throws SAXException
{
    System.out.println("Warning:"+e.getMessage());
}

public void error (SAXParseException e)
    throws SAXException
{
    throw new SAXException(e.getMessage());
}

public void fatalError (SAXParseException e)
    throws SAXException
{
    System.out.println("Fatal error");
    throw new SAXException(e.getMessage());
}
}

```

Using XML Parser for Java: XSLT Processor

To implement the XSLT Processor in the XML Parser for Java use `XSLProcessor` class.

[Figure 20-6](#) shows the overall process used by class, `XSLProcessor`.

1. A new `XSLProcessor()` class declaration begins the XSLT process.
2. There are two inputs:
 - **“Stylesheet”**. First a stylesheet is built. A new `XSLStyleSheet()` class is declared with any of the following available methods:

- * `removeParam()`
- * `resetParam()`
- * `setParam()`
- **“XML input”**. This can repeat 1 through n times for a particular stylesheet. This inputs the “Process Stylesheet” step.

Both inputs can be one of four types:

- input stream
 - URL
 - XML document
 - Reader
3. The resulting stylesheet object and the XML input, feed the “Process Stylesheet” step, namely:

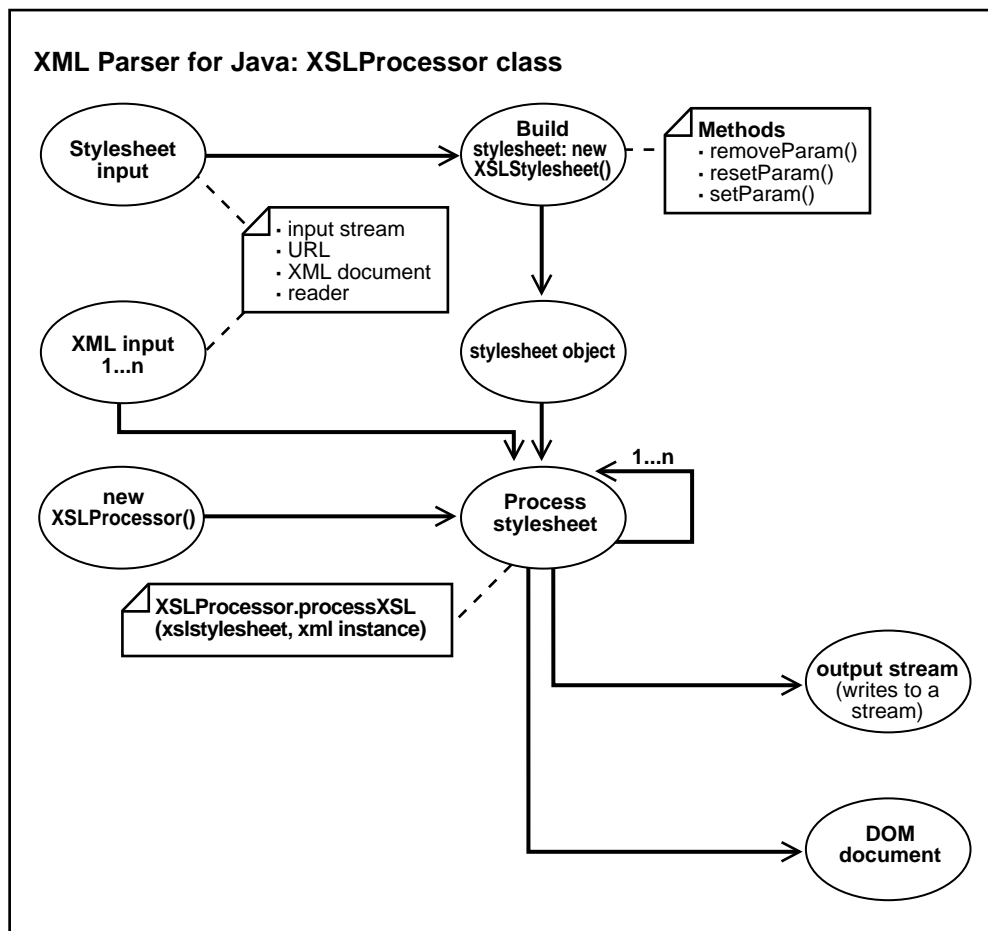
```
XSLProcessor.processXSL(xslstylesheet, xml instance)
```

4. The `XSLProcessor.processXSL()` method processes the XML input 1 through n times, using the selected stylesheet.
5. `XSLProcessor.processXSL()` outputs either an output stream or a DOM document.

XML Parser for Java XSLT Processor is illustrated by the following examples:

- ["XML Parser for Java Example 4: \(XSLSample.java\)"](#)
- ["XML Parser for Java Example 5: Using the DOM API and XSLT Processor"](#)

Figure 20–6 XSLProcessor Class Process



XML Parser for Java Example 4: (XSLSample.java)

```

/**
 * This file gives a simple example of how to use the XSL processing
 * capabilities of the Oracle XML Parser V2.0. An input XML document is
 * transformed using a given input stylesheet
 */

import org.w3c.dom.*;

```

```
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;

public class XSLSample
{
    /**
     * Transforms an xml document using a stylesheet
     * @param args input xml and xml documents
     */
    public static void main (String args[]) throws Exception
    {
        DOMParser parser;
        XMLDocument xml, xslDoc, out;
        URL xslURL;
        URL xmlURL;

        try
        {

            if (args.length != 2)
            {
                // Must pass in the names of the XSL and XML files
                System.err.println("Usage: java XSLSample xslfile xmlfile");
                System.exit(1);
            }

            // Parse xsl and xml documents

            parser = new DOMParser();
            parser.setPreserveWhitespace(true);

            // parser input XSL file
            xslURL = createURL(args[0]);
            parser.parse(xslURL);
            xslDoc = parser.getDocument();

            // parser input XML file
            xmlURL = createURL(args[1]);
            parser.parse(xmlURL);
            xml = parser.getDocument();

            // instantiate a stylesheet
            XSLStyleSheet xsl = new XSLStyleSheet(xslDoc, xslURL);
```

```
XSLProcessor processor = new XSLProcessor();

// display any warnings that may occur
processor.showWarnings(true);
processor.setErrorStream(System.err);

// Process XSL
DocumentFragment result = processor.processXSL(xsl, xml);

// create an output document to hold the result
out = new XMLDocument();

// create a dummy document element for the output document
Element root = out.createElement("root");
out.appendChild(root);

// append the transformed tree to the dummy document element
root.appendChild(result);

// print the transformed document
out.print(System.out);
}
catch (Exception e)
{
    e.printStackTrace();
}
}

// Helper method to create a URL from a file name
static URL createURL(String fileName)
{
    URL url = null;
    try
    {
        url = new URL(fileName);
    }
    catch (MalformedURLException ex)
    {
        File f = new File(fileName);
        try
        {
            String path = f.getAbsolutePath();
            // This is a bunch of weird code that is required to
            // make a valid URL on the Windows platform, due
            // to inconsistencies in what getAbsolutePath returns.

```

```
String fs = System.getProperty("file.separator");
if (fs.length() == 1)
{
    char sep = fs.charAt(0);
    if (sep != '/')
        path = path.replace(sep, '/');
    if (path.charAt(0) != '/')
        path = '/' + path;
}
path = "file://" + path;
url = new URL(path);
}
catch (MalformedURLException e)
{
    System.out.println("Cannot create url for: " + fileName);
    System.exit(0);
}
}
return url;
}
}
```

XML Parser for Java Example 5: Using the DOM API and XSLT Processor

This example code is not included in the `sample/` subdirectory. It uses the XML Parser for Java v2, to perform the following tasks.

- Parse an XML document
- Use the DOM API, to manipulate the XML data
- Use the XSLT Processor to transform the data

```
import org.w3c.dom.*;
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;
public class XSLTransform
{
    public static void main (String args[]) throws Exception
    {
        DOMParser parser;
        XMLDocument xml, xslDoc, out;
```

```
URL xslURL;
URL xmlURL;

try
{
    if (args.length != 2)
    {
        // Pass in the names of the XSL and XML files
        System.err.println("Usage: java XSLTransform
            xslfile xmlfile");
        System.exit(1);
    }

    // Parse XSL and XML documents
    parser = new DOMParser();
    parser.setPreserveWhitespace(true);

    xslURL = createURL(args[0]);
    parser.parse(xslURL);
    xslDoc = parser.getDocument();

    xmlURL = createURL(args[1]);
    parser.parse(xmlURL);
    xml = parser.getDocument();

    // Instantiate the stylesheet
    XSLStyleSheet xsl = new XSLStyleSheet(xslDoc, xslURL);

    XSLProcessor processor = new XSLProcessor();

    // Display any warnings that may occur
    processor.showWarnings(true);
    processor.setErrorStream(System.err);

    // Process XSL
    DocumentFragment result = processor.processXSL(xsl, xml);

    // Create an output document to hold the result
    out = new XMLDocument();

    // Create a dummy document element for the output document
    Element root = out.createElement("root");
    out.appendChild(root);

    // Append the transformed tree to the dummy document element
```

```
        root.appendChild(result);

        // Print the transformed document
        out.print(System.out);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

Comments on XSLT Example 5

See [Figure 20-4](#) and [Figure 20-6](#). The following provides comments for Example 5:

1. The program inputs two URL documents:
 - URL xmlURL;
 - URL xslURL;
2. Parse the two documents and set the preserve white space property:

```
parser = new DOMParser();
parser.setPreserveWhitespace(true);
```

3. Get the XSL and XML documents

```
xslURL = createURL(args[0]);
parser.parse(xslURL);
xslDoc = parser.getDocument();

xmlURL = createURL(args[1]);
xmlURL = createURL(args[1]);
parser.parse(xmlURL);
xml = parser.getDocument();
```

4. Initialize a new XSLStylesheet and XSLProcessor class:

```
XSLStylesheet xsl = new XSLStylesheet(xslDoc, xslURL);

XSLProcessor processor = new XSLProcessor();
processor.setErrorStream(System.err);
```

5. Process the stylesheet

```
DocumentFragment result = processor.processXSL(xsl, xml);
```

6. Output the DOM XML transformed document

```
out = new XMLDocument();
Element root = out.createElement("root");
out.appendChild(root);
root.appendChild(result);
```

Using XML Parser for Java: SAXNamespace() Class

Using the SAXNamespace() class is illustrated in the following example:

- ["XML Parser for Java Example 6: \(SAXNamespace.java\)"](#)

XML Parser for Java Example 6: (SAXNamespace.java)

```
// This file demonstrates a simple use of the Namespace extensions to
// the SAX APIs.

import org.xml.sax.*;
import java.io.*;
import java.net.URL;
import java.net.MalformedURLException;

// Extensions to the SAX Interfaces for Namespace support.
import oracle.xml.parser.v2.XMLDocumentHandler;
import oracle.xml.parser.v2.DefaultXMLDocumentHandler;
import oracle.xml.parser.v2.NSName;
import oracle.xml.parser.v2.SAXAttrList;

import oracle.xml.parser.v2.SAXParser;

public class SAXNamespace {
    static public void main(String[] args) {
        String fileName;

        //Get the file name
        if (args.length == 0)
        {
            System.err.println("No file Specified!!!");
            System.err.println("USAGE: java SAXNamespace <filename>");
            return;
        }
    }
}
```

```
else
{
    fileName = args[0];
}

try {
    // Create handlers for the parser
    // Use the XMLDocumentHandler interface for namespace support
    // instead of org.xml.sax.DocumentHandler
    XMLDocumentHandler xmlDocHandler = new XMLDocumentHandlerImpl();

    // For all the other interface use the default provided by
    // Handler base
    HandlerBase defHandler = new HandlerBase();

    // Get an instance of the parser
    SAXParser parser = new SAXParser();

    // Set Handlers in the parser
    // Set the DocumentHandler to XMLDocumentHandler
    parser.setDocumentHandler(xmlDocHandler);

    // Set the other Handler to the defHandler
    parser.setErrorHandler(defHandler);
    parser.setEntityResolver(defHandler);
    parser.setDTDHandler(defHandler);

    try
    {
        parser.parse(fileToURL(new File(fileName)).toString());
    }
    catch (SAXParseException e)
    {
        System.err.println(args[0] + ": " + e.getMessage());
    }
    catch (SAXException e)
    {
        System.err.println(args[0] + ": " + e.getMessage());
    }
}
catch (Exception e)
{
    System.err.println(e.toString());
}
}
```



```

static public URL fileToURL(File file)
{
    String path = file.getAbsolutePath();
    String fSep = System.getProperty("file.separator");
    if (fSep != null && fSep.length() == 1)
        path = path.replace(fSep.charAt(0), '/');
    if (path.length() > 0 && path.charAt(0) != '/')
        path = '/' + path;
    try {
        return new URL("file", null, path);
    }
    catch (java.net.MalformedURLException e) {
        /* According to the spec this could only happen if the file
        protocol were not recognized. */
        throw new Error("unexpected MalformedURLException");
    }
}

private SAXNamespace() throws IOException
{
}

}

/*****
Implementation of XMLDocumentHandler interface. Only the new
startElement and endElement interfaces are implemented here. All other
interfaces are implemented in the class HandlerBase.
*****/

class XMLDocumentHandlerImpl extends DefaultXMLDocumentHandler
{

    public void XMLDocumentHandlerImpl()
    {
    }

    public void startElement(NSName name, SAXAttrList atts) throws SAXException
    {

        // Use the methods getQualifiedName(), getLocalName(), getNamespace()
        // and getExpandedName() in NSName interface to get Namespace
        // information.
        String qName;

```

```
String localName;
String nsName;
String expName;
qName = name.getQualifiedName();
System.out.println("ELEMENT Qualified Name:" + qName);
localName = name.getLocalName();
System.out.println("ELEMENT Local Name      :" + localName);

nsName = name.getNamespace();
System.out.println("ELEMENT Namespace      :" + nsName);

expName = name.getExpandedName();
System.out.println("ELEMENT Expanded Name  :" + expName);

for (int i=0; i<atts.getLength(); i++)
{

// Use the methods getQualifiedName(), getLocalName(), getNamespace()
// and getExpandedName() in SAXAttrList interface to get Namespace
// information.
qName = atts.getQualifiedName(i);
localName = atts.getLocalName(i);
nsName = atts.getNamespace(i);
expName = atts.getExpandedName(i);

System.out.println(" ATTRIBUTE Qualified Name  :" + qName);
System.out.println(" ATTRIBUTE Local Name      :" + localName);
System.out.println(" ATTRIBUTE Namespace      :" + nsName);
System.out.println(" ATTRIBUTE Expanded Name  :" + expName);

// You can get the type and value of the attributes either
// by index or by the Qualified Name.
String type = atts.getType(qName);
String value = atts.getValue(qName);

System.out.println(" ATTRIBUTE Type              :" + type);
System.out.println(" ATTRIBUTE Value            :" + value);
System.out.println();
}
}

public void endElement(NSName name) throws SAXException
{
// Use the methods getQualifiedName(), getLocalName(), getNamespace()
// and getExpandedName() in NSName interface to get Namespace
```

```

        // information.
        String expName = name.getExpandedName();
        System.out.println("ELEMENT Expanded Name : " + expName);
    }
}

```

XML Parser for Java: Command Line Interface

oraxml - Oracle XML parser

`oraxml` is a command-line interface to parse an XML document. It checks for well-formedness and validity.

To use `oraxml` ensure the following:

- Your CLASSPATH environment variable is set to point to the `xmlparserv2.jar` file that comes with Oracle XML V2 parser for Java.
- Your PATH environment variable can find the java interpreter that comes with JDK 1.1.x or JDK 1.2.
- Because `oraxml` supports schema validation, include `xschema.jar` also in your CLASSPATH

Use the following syntax to invoke `oraxml`:

```
oraxml options* source
```

`oraxml` expects to be given an XML file to parse. [Table 20–3](#) lists `oraxml`'s command line options.

Table 20–3 *oraxml: Command Line Options*

Option	Purpose
-h	Help mod. Prints <code>oraxml</code> invocation syntax.
-v	Partial validation mo. If this option is not used, the parser checks only for well formedness.
-s	Strict validation mode.
-w	Show warnings. By default, warnings are turned off.
-debug	Debug mod. By default, debug mode is turned off.

Table 20–3 oraxml: Command Line Options

Option	Purpose
-e <error log>	A file to write errors to. Specify a log file to write errors and warnings.

oraxsl - Oracle XSL processor

`oraxsl` is a command-line interface used to apply a stylesheet on multiple XML documents. It accepts a number of command-line options that dictate how it should behave.

To use `oraxsl` ensure the following:

- Your CLASSPATH environment variable is set to point to the `xmlparserv2.jar` file that comes with Oracle XML V2 parser for Java.
- Your PATH environment variable can find the java interpreter that comes with JDK 1.1.x or JDK 1.2.

Use the following syntax to invoke `oraxsl`:

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

`oraxsl` expects to be given a stylesheet, an XML file to transform, and optionally, a result file. If no result file is specified, it outputs the transformed document to standard out. If multiple XML documents need to be transformed by a stylesheet, the `-l` or `-d` options in conjunction with the `-s` and `-r` options should be used instead. These and other options are described in [Table 20–4](#).

Table 20–4 oraxsl: Command Line Options

Option	Purpose
-h	Help mode (prints oraxsl invocation syntax)
-v	Verbose mode (some debugging information is printed and could help in tracing any problems that are encountered during processing)
-w	Show warnings (by default, warnings are turned off)
-debug	New - Debug mode (by default, debug mode is turned off)
-e <error log>	A file to write errors to (specify a log file to write errors and warnings).

Table 20–4 oraxsl: Command Line Options (Cont.)

Option	Purpose
-t <# of threads>	Number of threads to use for processing (using multiple threads could provide performance improvements when processing multiple documents).
-l <xml file list>	List of files to transform (allows you to explicitly list the files to be processed).
-d <directory>	Directory with files to transform (the default behavior is to process all files in the directory). If only a certain subset of the files in that directory, e.g., one file, need to be processed, this behavior must be changed by using -l and specifying just the files that need to be processed. You could also change the behavior by using the '-x' or '-i' option to select files based on their extension).
-x <source extension>	Extensions to exclude (used in conjunction with -d. All files with the specified extension will not be selected).
-i <source extension>	Extensions to include (used in conjunction with -d. Only files with the specified extension will be selected).
-s <stylesheet>	Stylesheet to use (if -d or -l is specified, this option needs to be specified to specify the stylesheet to be used. The complete path must be specified).
-r <result extension>	Extension to use for results (if -d or -l is specified, this option must be specified to specify the extension to be used for the results of the transformation. So, if one specifies the extension "out", an input document "foo" would get transformed to "foo.out". By default, the results are placed in the current directory. This is can be changed by using the -o option which allows you to specify a directory to hold the results).
-o <result directory>	Directory to place results (this must be used in conjunction with the -r option).
-p	List of Parameters

XML Extension Functions for XSLT Processing

XSLT Processor Extension Functions: Introduction

XML extension functions for XSLT processing allow users of XSLT processor to call any Java method from XSL expressions. Java extension functions should belong to the namespace that starts with the following:

`http://www.oracle.com/XSL/Transform/java/`

An extension function that belongs to the following namespace:

`http://www.oracle.com/XSL/Transform/java/classname`

refers to methods in class `classname`. For example, the following namespace:

`http://www.oracle.com/XSL/Transform/java/java.lang.String`

can be used to call `java.lang.String` methods from XSL expressions.

Static Versus Non-static Methods

If the method is a non-static method of the class, then the first parameter will be used as the instance on which the method is invoked, and the rest of the parameters are passed on to the method.

If the extension function is a static method, then all the parameters of the extension function are passed on as parameters to the static function.

XML Parser for Java - XSL Example 1: Static function

The following XSL, static function example:

```
<xsl:stylesheet
xmlns:math="http://www.oracle.com/XSL/Transform/java/java.lang.Math">
  <xsl:template match="/">
    <xsl:value-of select="math:ceil('12.34')"/>
  </xsl:template>
</xsl:stylesheet>
```

prints out '13'.

Constructor Extension Function

The extension function 'new' creates a new instance of the class and acts as the constructor.

XML Parser for Java - XSL Example 2: Constructor Extension Function

The following constructor function example:

```
<xsl:stylesheet
xmlns:jstring="http://www.oracle.com/XSL/Transform/java/java.lang.String">
  <xsl:template match="/">
```

```

<!-- creates a new java.lang.String and stores it in the variable str1 -->
<xsl:variable name="str1" select="jstring:new('Hello World')"/>
<xsl:value-of select="jstring:toUpperCase($str1)"/>
</xsl:template>
</xsl:stylesheet>

```

prints out 'HELLO WORLD'.

Return Value Extension Function

The result of an extension function can be of any type, including the five types defined in XSL:

- NodeList
- boolean
- String
- Number
- resulttree

They can be stored in variables or passed onto other extension functions.

If the result is of one of the five types defined in XSL, then the result can be returned as the result of an XSL expression.

XML Parser for Java XSL- XSL Example 3: Return Value Extension Function

Here is an XSL example illustrating the Return value extension function:

```

<!-- Declare extension function namespace -->
<xsl:stylesheet xmlns:parser =
"http://www.oracle.com/XSL/Transform/java/oracle.xml.parser.v2.DOMParser"
xmlns:document =
"http://www.oracle.com/XSL/Transform/java/oracle.xml.parser.v2.XMLDocument" >

<xsl:template match = "/"> <!-- Create a new instance of the parser, store it in
myparser variable -->
<xsl:variable name="myparser" select="parser:new()"/>
<!-- Call a non-static method of DOMParser. Since the method is anon-static
method, the first parameter is the instance on which the method is called. This
is equivalent to $myparser.parse('test.xml') -->
<xsl:value-of select="parser:parse($myparser, 'test.xml')"/>
<!-- Get the document node of the XML Dom tree -->
<xsl:variable name="mydocument" select="parser:getDocument($myparser)"/>

```

```
<!-- Invoke getelementsbytagname on mydocument -->
<xsl:for-each select="document:getElementsByTagName($mydocument, 'elementname')">
.....
</xsl:for-each> </xsl:template>
</xsl:stylesheet>
```

Datatypes Extension Function

Overloading based on number of parameters and type is supported. Implicit type conversion is done between the five XSL types as defined in XSL.

Type conversion is done implicitly between (String, Number, Boolean, ResultTree) and from NodeSet to (String, Number, Boolean, ResultTree).

Overloading based on two types which can be implicitly converted to each other is not permitted.

XML Parser for Java - XSL Example 4: Datatype Extension Function

The following overloading will result in an error in XSL, since String and Number can be implicitly converted to each other:

- `abc(int i){}`
- `abc(String s){}`

Mapping between XSL type and Java type is done as following:

```
String -> java.lang.String
Number -> int, float, double
Boolean -> boolean
NodeSet -> XMLNodeList
ResultTree -> XMLDocumentFragment
```

ora XSLT Built In Extensions: ora:node-set and ora:output

The following example illustrates both ora:node-set and ora:output in action.

If you do:

```
$ oraxsl foo.xml slides.xsl toc.html
```

where "foo.xml" is any XML file, you get:

- A "toc.html" slide with a table of contents
- A "slide01.html" file with slide 1

- A "slide02.html" file with slide 2

```

<!--
ora:output
namespace
"http://www.oracle.com/XSL/Transform/java"
+-->
<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:ora="http://www.oracle.com/XSL/Transform/java">

result document -->
<!-- <xsl:output> affects the primary
<xsl:output mode="html" indent="no"/>

all attributes
must provide the
assign a name to
later.
indent="no"/>

result-tree fragment
<!--
| Illustrate using ora:node-set and
|
| Both extensions depend on defining a
| with the uri of
"http://www.oracle.com/XSL/Transform/java"
+-->
<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:ora="http://www.oracle.com/XSL/Transform/java">

<!-- <xsl:output> affects the primary
<xsl:output mode="html" indent="no"/>

<!--
| <ora:output> at the top-level allows
| that <xsl:output> allows, but you
| additional "name" attribute to
| these output settings to be used
+-->
<ora:output name="myOutput" mode="html"
indent="no"/>

<!--
| This top-level variable is a
+-->
<xsl:variable name="fragment">
<slides>
<slide>
<title>First Slide</title>
<bullet>Point One</bullet>
<bullet>Point Two</bullet>
<bullet>Point Three</bullet>
</slide>

```

```

        <slide>
            <title>Second Slide</title>
            <bullet>Point One</bullet>
            <bullet>Point Two</bullet>
            <bullet>Point Three</bullet>
        </slide>
    </slides>
</xsl:variable>
<xsl:template match="/">
    <!--
    | We cannot "de-reference" a
    |
    | navigate into it with an XPath
    |
    | the ora:node-set() built-in
    |
    | "cast" a result-tree fragment to a
    |
    | then be navigated using XPath. Since
    |
    | of <slides> twice below, we save the
    |
    +-->
    <xsl:variable name="slides"
    <!--
    | This <html> page will go to the
    |
    | It is a "table of contents" for the
    |
    | links to each slide. The "slides"
    |
    | into *secondary* result documents,
    |
    | a file name of "slideNN.html" where
    |
    | slide number
    +-->
    <html>
        <body>
            <h1>List of All Slides</h1>
            <xsl:apply-templates
    <!--
    |
    |
    +-->
    </body>
</html>

```

result-tree-fragment to
expression. However, using
extension function, you can
node-set which *can*
we'll use the node-set
node-set in a variable.

```

select="ora:node-set($fragment)"/>

```

primary result document.
slide show, with
will each be generated
each slide having
NN is the two-digit

```

select="$slides" mode="toc"/>

```

```

each slide
    <!--
    | Now go apply-templates to format
    +-->
    <xsl:apply-templates select="$slides"/>
</xsl:template>
<!-- In 'toc' mode, generate a link to
each slide we match -->
    <xsl:template match="slide" mode="toc">
    <a
href="slide{format-number(position(),'00')}.html">
        <xsl:value-of select="title"/>
    </a><br/>
    </xsl:template>
<!--
    | For each slide matched, send the
output for the current
    | <slide> to a file named
"slideNN.html". Use the named
    | output style defined above called
"myOutput".
    +-->
    <xsl:template match="slide">
    <ora:output use="myOutput"
href="slide{format-number(position(),'00')}.html">
        <html>
        <body>
            <xsl:apply-templates
select="title"/>
            <ul>
                <xsl:apply-templates
select="*[not(self::title)]"/>
            </ul>
        </body>
        </html>
    </ora:output>
    </xsl:template>
<xsl:template match="bullet">
    <li><xsl:value-of select="."/></li>
</xsl:template>
<xsl:template match="title">
    <h1><xsl:value-of select="."/></h1>
</xsl:template>
</xsl:stylesheet>

```


Frequently Asked Questions (FAQs): XML Parser for Java

The XML Parser for Java Frequently Asked Questions (FAQs) are organized into the following topics:

- [DTDs](#)
- [DOM and SAX APIs](#)
- [Validation](#)
- [Character Sets](#)
- [Adding XML Document as a Child](#)
- [Uninstalling Parsers](#)
- [XML Parser for Java: Installation](#)
- [General XML Parser Related Questions](#)
- [XSLT Processor and XSL Stylesheets](#)
- [Compressing Large Volumes of XML Documents](#)

DTDs

Checking DTD Syntax: Suggestions for Editors

Question

I was wondering if someone could help me verify the syntax for the following DTD. I realize that I can use a DTD editor to do this for me, but the editor I'm using is not very good.

```
<?xml version="1.0"?>
<!DOCTYPE CATALOG [

<!ELEMENT CATALOG ( ADMIN, SCHEMA?, DATA? ) >
<!ATTLIST CATALOG xml:lang NMTOKEN #IMPLIED >

<!ELEMENT ADMIN ( NAME, INFORMATION) >
<!ELEMENT SCHEMA (CATEGORY | DESCRIPTOR)* >
<!ELEMENT DATA (ITEM)*>

<!ELEMENT NAME (#PCDATA) >
<!ELEMENT INFORMATION ( DATE, SOURCE ) >
```

```

<!ELEMENT DATE (#PCDATA) >
<!ELEMENT SOURCE (#PCDATA) >

<!ELEMENT CATEGORY (NAME | KEY | TYPE | UPDATE )* >
<!ATTLIST CATEGORY ACTION (ADD|DELETE|UPDATE) #REQUIRED>
<!ELEMENT DESCRIPTOR (NAME | KEY | UPDATE | OWNER | TYPE )* >
<!ATTLIST DESCRIPTOR ACTION (ADD|DELETE|UPDATE) #REQUIRED>
<!ELEMENT OWNER (NAME?, KEY? ) >
<!ELEMENT KEY (#PCDATA) >
<!ELEMENT TYPE (#PCDATA) >

<!ELEMENT ITEM (OWNER?, NAMEVALUE*, UPDATE ) >
<!ATTLIST ITEM ACTION (ADD | DELETE | UPDATE) #REQUIRED>
<!ELEMENT UPDATE (NAME | KEY | NAMEVALUE )* >

<!ELEMENT NAMEVALUE ( NAME, VALUE ) >
<!ELEMENT VALUE (#PCDATA)* >
]>

```

I'm unsure about the ATTLIST syntax.

Answer

I loaded this into XMLAuthority 1.1 and did a Save As. XML Authority lets you visually inspect and edit DTD's and XML Schemas. Highly recommended. <http://www.extensibility.com> (\$99.00).

It came back with:

```

<!ELEMENT CATALOG (ADMIN , SCHEMA? , DATA? )>
<!ATTLIST CATALOG xml:lang NMTOKEN #IMPLIED >
<!ELEMENT ADMIN (NAME , INFORMATION )>
<!ELEMENT SCHEMA (CATEGORY | DESCRIPTOR )*>
<!ELEMENT DATA (ITEM )*>
<!ELEMENT NAME (#PCDATA )>
<!ELEMENT INFORMATION (DATE , SOURCE )>
<!ELEMENT DATE (#PCDATA )>
<!ELEMENT SOURCE (#PCDATA )>
<!ELEMENT CATEGORY (NAME | KEY | TYPE | UPDATE )*>
<!ATTLIST CATEGORY ACTION (ADD | DELETE | UPDATE ) #REQUIRED >
<!ELEMENT DESCRIPTOR (NAME | KEY | UPDATE | OWNER | TYPE )*>
<!ATTLIST DESCRIPTOR ACTION (ADD | DELETE | UPDATE ) #REQUIRED >
<!ELEMENT OWNER (NAME? , KEY? )>
<!ELEMENT KEY (#PCDATA )>
<!ELEMENT TYPE (#PCDATA )>

```

```
<!ELEMENT ITEM (OWNER? , NAMEVALUE* , UPDATE )>
<!ATTLIST ITEM ACTION (ADD | DELETE | UPDATE ) #REQUIRED >
<!ELEMENT UPDATE (NAME | KEY | NAMEVALUE )*>
<!ELEMENT NAMEVALUE (NAME , VALUE )>
<!ELEMENT VALUE (#PCDATA )*>
```

DTD File in DOCTYPE Must be Relative to XML Document Location

Question

My parser doesn't find the DTD file.

Answer

The DTD file defined in the `<!DOCTYPE>` declaration must be relative to the location of the input XML document. Otherwise, you'll need to use the `setBaseURL(url)` functions to set the base URL to resolve the relative address of the DTD if the input is coming from an `InputStream`.

Validating an XML File Using External DTD

Question

Can I validate an XML file using an external DTD?

Answer

You need to include a reference to the applicable DTD in your XML document. Without it there is no way that the parser knows what to validate against. Including the reference is the XML standard way of specifying an external DTD. Otherwise you need to embed the DTD in your XML Document.

DTD Caching

Question

Do you have DTD caching? How do I set the DTD using v2 parser for DTD Cache purpose?

Answer

Yes, DTD caching is optional and is not enabled automatically.

The method to set the DTD is `setDoctype()`. Here is an example:

```
// Test using InputSource
parser = new DOMParser();
parser.setErrorStream(System.out);
parser.showWarnings(true);

FileReader r = new FileReader(args[0]);
InputSource inSource = new InputSource(r);
inSource.setSystemId(createURL(args[0]).toString());
parser.parseDTD(inSource, args[1]);
dtd = (DTD)parser.getDoctype();

r = new FileReader(args[2]);
inSource = new InputSource(r);
inSource.setSystemId(createURL(args[2]).toString());
parser.setDoctype(dtd);
parser.setValidationMode(DTD_validation);
parser.parse(inSource);

doc = (XMLDocument)parser.getDocument();
doc.print(new PrintWriter(System.out));
```

Recognizing External DTDs

Question

How can XML Parser for Java (V2) recognize external DTD's when running from the server. The Java code has been loaded with `loadjava` and runs in the Oracle server process. My XML file has an external DTD reference.

1. But is there a more generic way, as with the SAX parser, to redirect it to a stream or string or something if my DTD is in the database?
2. Do you have a more generic way to redirect the DTD, analogous to that offered by the SAXParser with `resolveEntity()`.

Answer

1. We only have the `setBaseURL()` method at this time.
2. You can achieve your desired result using the following:
 - a. Parse your External DTD using a DOMParser's `parseDTD()` method.
 - b. Call `getDoctype()` to get an instance of `oracle.xml.parser.v2.DTD`

- c. On the document where you want to set your DTD programmatically, use the: `setDoctype(yourDTD)`; We use this technique to read a DTD out of our product's JAR file.

Loading external DTD's from a jar File

Question

I would like to put all my DTDs in a jar file, so that when the XML Parser needs a DTD it can get it from the jar. The current XML Parser supports a base URL(`setBaseURL()`), but that just points to a place where all the DTDs are exposed.

Answer

The solution involves a combination of:

1. Load DTD as `InputStream` using:

```
InputStream is =  
YourClass.class.getResourceAsStream( "/foo/bar/your.dtd" );  
This will open ./foo/bar/your.dtd in the first relative location on the  
CLASSPATH that it can be found, including out of your jar if it's in the  
CLASSPATH.
```

2. Parse the DTD with the code:

```
DOMParser d = new DOMParser();  
d.parseDTD(is, "rootelementname");  
d.setDoctype(d.getDoctype());
```

3. Now parse your document with:

```
d.parse("yourdoc");
```

Can I Check the Correctness of an XML Document Using their DTD?

Question

I am exporting Java objects to XML. I can construct a DOM with an XML Document and use its `print` method to export it. But, I am unable to set the DTD of these documents. I construct a parser, parse the DTD, and then get the DTD via `Document doc = parser.getDocument()` and `DocType dtd = doc.getDocumentType()`.

How do I set the DTD of the freshly constructed XML Documents to use this one in order to be able to check the correctness of the documents using this DTD at a later time?

Answer

Your method of getting the DTD object is correct. However, we do not do any validation while creating the DOM tree using DOM APIs. So setting the DTD in the Document will not help validate the DOM tree that is constructed. The only way to validate an XML file is to parse the XML document using DOMParser or SAXParser.

Parsing a DTD Object Separately from XML Document

Question

How do I parse and get a DTD Object separately from parsing my XML document?

Answer

The `parseDTD()` method allows you to parse a DTD file separately and get a DTD object. Here is a sample code to do that:

```
DOMParser domparser = new DOMParser();
domparser.setValidationMode(DTD_validation);
/* parse the DTD file */
domparser.parseDTD(new FileReader(dtdfile));
DTD dtd = domparser.getDocType();
```

Case-Sensitivity in Parser Validation against DTD?

Question

The XML file has a tag like: `<xn:subjectcode>`. In the DTD, it is defined as `<xn:subjectCode>`. When the file is parsed and validated against the DTD, it gives an error: XML-0148: (Error) Invalid element 'xn:subjectcode' in content of 'xn:Resource',...

When I changed the element name to `<xn:subjectCode>` instead of `<xn:subjectcode>` it works. Is the parser case-sensitive as far as validation against DTD's go - or is it because, there is a namespace also in the tag definition of the element and when a element is defined along with its namespace, the case-sensitivity comes into effect?

Answer

XML is inherently case-sensitive, therefore our parsers enforce case sensitivity in order to be compliant. When you run in non-validation mode only well-formedness counts. However `<test></Test>` would signal an error even in non-validation mode.

Extracting Embedded XML From a CDATA Section

Question

1. I want to extract PAYLOAD and do extra processing on it.
2. When I select the value of PAYLOAD it does not parse the data because it is in a CDATA section.
3. How do I extract embedded XML using just XSLT. I have done this using SAX before but in the current setup all I can use is XSLT.

Answer

1. Here are the answers:

```
<PAYLOAD>
<![CDATA[<?xml version = '1.0' encoding = 'ASCII' standalone = 'no'?>
<ADD_PO_003>
  <CNTR0LAREA>
    <BSR>
      <VERB value="ADD">ADD</VERB>
      <NOUN value="PO">PO</NOUN>
      <REVISION value="003">003</REVISION>
    </BSR>
  </CNTR0LAREA>
</ADD_PO_003>]]>
</PAYLOAD>
```

The CDATA strategy is kind of odd. You won't be able to use a different encoding on the nested XML document included as text inside the CDATA, so having the XML Declaration of the embedded document seems of little value to me. If you don't need the XML Declaration, then why not just embed the message as real elements into the `<PAYLOAD>` instead of as a text chunk which is what CDATA does for you.

Just do:

```
String s = YourDocumentObject.selectSingleNode("/OES_MESSAGE/PAYLOAD");
```

- It shouldn't parse the data, you've asked for it to be a big text chunk, which is what it will give you. You'll have to parse the text chunk yourself (another benefit of not using the CDATA approach) by doing something like:

```
YourParser.parse( new StringReader(s));
```

where *s* is the string you got in the previous step.

- There's nothing special about what's in your CDATA, it's just text. If you want the text content to be output without escaping the angle-brackets, then you'll do:

```
<xsl:value-of select="/OES_MESSAGE/PAYLOAD" disable-output-escaping="yes"/>
```

Why Am I Getting an Error When I Call DOMParser.parseDTD()?

Question

I am having trouble creating a DTD and parsing it using Oracle XML Parser for Java v2. I got the following error when I call DOMParser.parseDTD() function:

Attribute value should start with quote.

Please check my DTD and tell me what's wrong?

```
<?xml version = "1.0" encoding="UTF-8" ?>
<!-- RCS_ID = "$Header: XMLRenderer.dtd 115.0 2000/09/18 03:00:10 fli noship $"
-->
<!-- RCS_ID_RECORDERED = VersionInfo.recordClassVersion(RCS_ID,
"oracle.apps.mwa.admin") -->
<!-- Copyright: This DTD file is owned by Oracle Mobile Application Server
Group. -->
<!ELEMENT page (header?,form,footer?) >
<!ATTLIST page
name CDATA #REQUIRED
lov (Y|N) 'N' >
<!ELEMENT header EMPTY >
<!ATTLIST header
name CDATA #REQUIRED
title CDATA
home (Y|N) 'N'
portal (Y|N) 'N'
logout (Y|N) 'N' >
<!ELEMENT footer EMPTY >
<!ATTLIST footer
```

```

        name    CDATA    #REQUIRED
        home    (Y|N)    'N'
        portal  (Y|N)    'N'
        logout  (Y|N)    'N'
        copyright (Y|N) 'N' >

<!ELEMENT    form
(styledText|textInput|list|link|menu|submitButton|table|separator)+ >
<!ATTLIST    form
        name    CDATA    #REQUIRED
        title   CDATA
        type    CDATA >

<!ELEMENT    styledText    (#PCDATA) >

<!ELEMENT    textInput    EMPTY >
<!ATTLIST    textInput
        name    CDATA    #REQUIRED
        prompt  CDATA    #IMPLIED
        password (Y|N)    'N'
        required (Y|N)    'N'
        maxlength #IMPLIED
        size     #IMPLIED
        format   #IMPLIED
        default  #IMPLIED >

<!ELEMENT    link (postfield*) >
<!ATTLIST    link
        name    CDATA    #REQUIRED
        title   CDATA    #REQUIRED
        baseurl CDATA    #REQUIRED >

```

Answer

Your DTD syntax is not valid. When you declare ATTLIST with CDATA, you must put #REQUIRED, #IMPLIED, #FIXED, “any value”, %paramatic_entity. For example, your DTD contains

```

<!ELEMENT    header    EMPTY >
<!ATTLIST    header
        name    CDATA    #REQUIRED
        title   CDATA
        home    (Y|N)    'N'
        portal  (Y|N)    'N'
        logout  (Y|N)    'N' >

```

should change as follows:

```
<!ELEMENT header EMPTY >
<!ATTLIST header
    name CDATA #REQUIRED
    title CDATA #REQUIRED <- can replaced by #FIXED, #IMPLIED, or
"titleLabel"
    home (Y|N) 'N'
    portal (Y|N) 'N'
    logout (Y|N) 'N' >
```

What Is Standard Extension for External Entities References in an XML Document?

Question

Is there a standard extension (other than .xml or .txt) that should be used for external entities which are being referenced in an XML document. These external entities are not complete XML files, but rather only part of an XML file, starting with the <![CDATA[. Mostly they contain HTML, or Javascript code, but may also contain just some plain text. As an example, the external entity is A.txt which is being referenced in the XML document B.xml.

A.txt:

```
<![CDATA[<!-- This is just an html comment -->]]>
```

B.xml:

```
<?xml version="1.0"?>
<!DOCTYPE B[
<!ENTITY htmlComment SYSTEM "A.txt">
]>

<B>
    &htmlComment;
</B>
```

Currently we are using .txt as an extension for all such entities, but need to change that, otherwise the translation team assumes that these files need to get translated, whereas they don't. Is there a standard extension that we should be using?

Answer

I marked up your DTD syntax in “red (bold)” in your DTD. The file extension for external entities is unimportant so you can change it to any convenient extension, including *no* extension.:-)

DOM and SAX APIs

Using the DOM API

Question

How do I get the number of elements in a particular tag using the parser?

Answer

You can use the `getElementsByTagName()` method that returns a `NodeList` of all descent elements with a given tag name. You can then find out the number of elements in that `NodeList` to determine the number of the elements in the particular tag.

How DOM Parser Works

Question

How does the XML DOM parser work?

Answer

The parser accepts an XML formatted document and constructs in memory a DOM tree based on its structure. It will then check whether the document is well-formed and optionally whether it complies with a DTD. It also provides methods to support DOM Level 1 and 2.

Creating a Node With Value to be Set Later

Question

How do I create a node whose value I can set later?

Answer

If you check the DOM spec referring to the table discussing the node type, you will find that if you are creating an element node, its `nodeValue` is to be null and hence cannot be set. However, you can create a text node and append it to the element node. You can put the value in the text node.

Traversing the XML Tree

Question

How to traverse the XML tree

Answer

You can traverse the tree by using the DOM API. Or alternately, you can use the `selectNodes()` method which takes XPath syntax to navigate through the XML document. `selectNodes()` is part of `oracle.xml.parser.v2.XMLNode`.

Extracting Elements from XML File

Question

How do I extract elements from the XML file?

Answer

If you're using DOM, the `getElementsByTagName()` method can be used to get all of the elements in the document.

Does a DTD Validate the DOM Tree?

Question

If I add a DTD to an XML Document, does it validate the DOM tree?

Answer

No, we do not do any validation while creating the DOM tree using the DOM APIs. So setting the DTD in the Document will not help in validating the DOM tree that is constructed. The only way to validate an XML file is to parse the XML document using the `DOMParser` or `SAXParser`.

First Child Node Element Value

Question

How do I efficiently obtain the value of first child node of the element without going through the DOM Tree?

Answer

If you do not need the entire tree, use the SAX interface to return the desired data. Since it is event-driven, it does not have to parse the whole document.

Creating DocType Node

Question

How do I create a DocType Node?

Answer

The only current way of creating a doctype node is by using the parseDTD functions. For example, emp.dtd has the following DTD:

```
<!ELEMENT employee (Name, Dept, Title)>
  <!ELEMENT Name (#PCDATA)>
  <!ELEMENT Dept (#PCDATA)>
  <!ELEMENT Title (#PCDATA)>
```

You can use the following code to create a doctype node:

```
parser.parseDTD(new FileInputStream(emp.dtd), "employee");
dtd = parser.getDocType();
```

XMLNode.selectNodes() Method

Question

How do I use the selectNodes() method in XMLNode class?

Answer

The selectNodes() method is used in XMLElement and XMLDocument nodes. This method is used to extract contents from the tree/subtree based on the select

patterns allowed by XSL. The optional second parameter of `selectNodes`, is used to resolve Namespace prefixes (return the expanded namespace URL given a prefix). `XMLElement` implements `NSResolver`, so it can be sent as the second parameter. `XMLElement` resolves the prefixes based on the input document. You can implement the `NSResolver` interface, if you need to override the namespace definitions. The following sample code uses `selectNodes`

```
public class SelectNodesTest {
    public static void main(String[] args) throws Exception {
        String pattern = "/family/member/text()";
        String file = args[0];

        if (args.length == 2)
            pattern = args[1];

        DOMParser dp = new DOMParser();

        dp.parse(createURL(file)); // Include createURL from DOMSample
        XMLDocument xd = dp.getDocument();
        XMLElement e = (XMLElement) xd.getDocumentElement();
        NodeList nl = e.selectNodes(pattern, e);
        for (int i = 0; i < nl.getLength(); i++) {
            System.out.println(nl.item(i).getNodeValue());
        }
    }
}

> java SelectNodesTest family.xml
Sarah
Bob
Joanne
Jim

> java SelectNodesTest family.xml //member/@memberid
m1
m2
m3
m4
```

Using SAX API to Get the Data Value

Question

I am using SAX to parse an XML document. How does it get the value of the data?

Answer

During a SAX parse the value of an element will be the concatenation of the characters reported from after the startElement event to before the corresponding endElement event is called.

SAXSample.java**Question**

Inside the SAXSample program, I did not see any line that explicitly calls setDocumentLocator and some other methods. However, these methods are 'run'. Can you explain when they are called and from where

Answer

SAX is a standard interface for event-based XML parsing. The parser reports parsing events directly through callback functions such as setDocumentLocator() and startDocument(). The application, in this case, the SAXSample, implements handlers to deal with the different events. Here is a good place to help you start learning about the event-driven API, SAX:
<http://www.megginson.com/SAX/index.html>

Does DOMParser implement Parser interface**Question**

Does the XML Parser DOMParser implement org.xml.sax.Parser interface at all? The documentation says it implements XML Constants and the API does not include that class at all.

Answer

You'll want oracle.xml.parser.v2.SAXParser to work with SAX and to have something that implements the org.xml.sax.Parser interface.

Creating a New Document Type Node Via DOM**Question**

I am trying to create a XML file on the fly. I use the NodeFactory to construct a document (createDocument()). I have then setStandalone("no") and

setVersion("1.0"). when I try to add a DOCTYPE node via appendChild(new XMLNode("test", Node.DOCUMENT_TYPE_NODE)), I get a ClassCastException. What is the mechanism to add a node of this type? I noticed that the NodeFactory did not have a mechanism for creating a DOCTYPE node.

Answer

There is no mechanism to create a new DOCUMENT_TYPE_NODE object via DOM APIs. The only way to get a DTD object is to parse the DTD file or the XML file using the DOMParser, and then use the getDocType() method.

Note that new XMLNode("test",Node.DOCUMENT_TYPE_NODE) does not create a DTDobject. It creates an XMLNode object with the type set to DOCUMENT_TYPE_NODE, which in fact should not be allowed. The ClassCastException is raised because appendChild expects a DTDobject (based on the type).

Also, we do not do any validation while creating the DOM tree using the DOM APIs. So setting the DTD in the Document will not help in validating the DOM tree that is constructed. The only way to validate an XML file is to parse the XML document using DOMParser or SAXParser.

Querying for First Child Node's Value of a Certain Tag

Question

I am using the XML Parser for Java v2. Given a XML document containing the following Calculus Math Jim Green Jack Mary Paul, I want to obtain the value of first child node of whose tag is. I could not find any method that can do that efficiently. The nearest match is method getElementsByTagName("Name"), which traverses the entire tree under.

Answer

Your best bet, if you do not need the entire tree, is to use the SAX interface to return the desired data. Since it is event driven it does not have to parse the whole document.

XML Document Generation From Data in Variables

Question

Is there an example of XML document generation starting from information contained in simple variables? An example would be: A client fills a Java form and wants to obtain an XML document containing the given data.

Answer

Here are two possible interpretations of your question and answers to both. Let's say you have two variables in Java:

```
String firstname = "Gianfranco";  
String lastname = "Pietraforte";
```

The two ways that come to mind first to get this into an XML document are as follows:

1. Make an XML document in a string and parse it.

```
String xml = "<person><first>"+firstname+"</first>"+  
            "<last>"+lastname+"</last></person>";  
  
DOMParser d = new DOMParser();  
d.parse( new StringReader(xml));  
Document xmldoc = d.getDocument();
```

2. Use DOM APIs to construct the document and "stitch" it together:

```
Document xmldoc = new XMLDocument();  
Element e1 = xmldoc.createElement("person");  
xmldoc.appendChild(e1);  
Element e2 = xmldoc.createElement("first");  
e1.appendChild(e2);  
Text t = xmldoc.createTextNode(firstname);  
e2.appendChild(t);  
// and so on
```

Printing Data in the Element Tags: DOM API

Question

Can you suggest how to get a print out using the DOM API in Java:

```
<name>macy</name>
```

I want to print out "macy". Don't know which class and what function to use. I was successful in printing "name" on to the console.

Answer

For DOM, you need to first realize that `<name>macy</name>` is actually an element named "name" with a child node (Text Node) of value "macy".

So, you can do the following:

```
String value = myElement.getFirstChild().getNodeValue();
```

Building XML Files from Hashtable Value Pairs

Question

We have a hash table of key value pairs, how do we build an XML file out of it using the DOM API? We have a hashtablekey = valuenam = georgezip = 20000. How do we build this?

```
<key>value</key><name>george</name><zip>20000</zip>'
```

Is there a utility to do it automatically?

Answer

1. Get the enumeration of keys from your hashtable
2. Loop while `enum.hasMoreElements()`
3. For each key in the enumeration, use the `createElement()` on DOM Document to create an element by the name of the key with a child text node with the value of the *value* of the hashtable entry for that key.

XML Parser for Java: wrong_document_err on Node.appendChild()

Question

I have a question regarding our XML parser (v2) implementation. Say if I have the following scenario:

```
Document doc1 = new XMLDocument();
Element element1 = doc1.createElement("foo");
Document doc2 = new XMLDocument();
```

```
Element element2 = doc2.createElement("bar");
element1.appendChild(element2);
```

My question is whether or not we should get a `DOMException` of `WRONG_DOCUMENT_ERR` on calling the `appendChild()` routine. This comes to my mind when I look at the `XSLSample.java` distributed with the `XMLparser (v2)`. Any feedback would be greatly appreciated.

Answer

Yes you should get this error, since the owner document of `element1` is `doc1` while that of `element2` is `doc2`. `AppendChild()` only works within a single tree and you are dealing with two different ones.

Question 2

In `XSLSample.java` that's shipped with `xmlparser v2`:

```
DocumentFragment result = processor.processXSL(xsl, xml);
// create an output document to hold the result
out = new XMLDocument();
// create a dummy document element for the output document
Element root = out.createElement("root");
out.appendChild(root);
// append the transformed tree to the dummy document element
root.appendChild(result);
```

Nodes `root` and `result` are created from different XML Documents, wouldn't this result in the `WRONG_DOCUMENT_ERR` when we try to append `result` to `root`?

Answer 2

This sample uses a document fragment that does not have a root node, therefore there are not two XML documents.

Question 3

When appending a document fragment to a node, only the child nodes of the document fragment (but not the document fragment itself) is inserted. Wouldn't the parser check the owner document of these child nodes?

Comment

A `DocumentFragment` shouldn't be bound to a 'root' node, since, by definition, a fragment could very well be just a list of nodes. The root node, if any, should be

considered a single child. That is, you could for example take all the lines of an Invoice document, and add them into an ProviderOrder document, without taking the invoice itself. How do we create a documentFragment without root? As the XSLT Processor does, so that we can append it to other documents?

Creating Nodes: DOMException when Setting Node Value

Question

I get the following error:

oracle.xml.parser.DOMException: Node cannot be modified while trying to set the value of a newly created node as below:

```
String eName="Mynode";
XMLNode aNode = new XMLNode(eName, Node.ELEMENT_NODE);
aNode.setNodeValue(eValue);
```

How do I create a node whose value I can set later on?

Answer

Check the DOM notes where they discuss the node type. You will see that if you are creating an element node, its nodeValue is null and hence cannot be set.

With SAX, How Can I Force the Parser to Not Discard Whitespace?

Question

I receive the following error when reading the attached file using SAX (Oracle XML Parser, v.2.0.2.9.0): if character data starts with a whitespace, characters() method discards characters that follows whitespace.

Is this a bug or can I force the parser to not discard those characters?

Answer

Use XMLParser.setPreserveWhitespace(true) to force the parser to not discard whitespace.

Validation

DTD: Understanding DOCTYPE and Validating Parser

Question

I have an XML string contains the following reference to a DTD, that is physically located in the directory where I start my program. The validating XML parser complains that this file can not be found.

```
<!DOCTYPE xyz SYSTEM "xyz.dtd" >
```

What are the rules for locating DTDs on the disk? Can anyone point me to a decent discussion of DOCTYPE attribute descriptions.

Answer

Are you parsing an InputStream or a URL? If you are parsing an InputStream the parser doesn't know where that InputStream came from so it cannot find the DTD in the "same directory as the current file". The solution is to setBaseURL() on DOMParser() to give the parser the URL "hint" information to be able to derive the rest when it goes to get the DTD.

Can Multiple Threads Use Single XSLProcessor/Stylesheet?

Question

Can multiple threads use a single XSLProcessor/XSLStylesheet instance to perform concurrent (at the same time) transformations?

Answer

As long as you are processing multiple files with no more than one XSLProcessor/XSLStylesheet instance per XML file you can do this simultaneously using threads. If you take a look at the readme.html file in the bin directory, it describes ORAXSL which has a threads parameter for multi-threaded processing.

Is it Safe to Use Document Clones in Multiple Threads?

Question

Is it safe to use clones of a document in multiple threads? Is the public void `setParam(String,String)` throws `XSLException` method of Class `oracle.xml.parser.v2.XSLStylesheet` supported? If no, is there another way to pass parameters at runtime to the XSLT Processor?

Answer

If you are copying the global area set up by the constructor to another thread then it should work.

That method is supported since XML Parser release 2.0.2.5.

Comment

You have it in your docs, but it is not implemented in the `XSLStylesheet` class (windows zip edition). First update your zip download file.

```
public static void serve(Document template, Document data,Element
userdata,PrintWriter out)
{
    XMLDocument clone = (XMLDocument)data.cloneNode(true);
    clone.getDocumentElement().appendChild(userdata.cloneNode(true));
    serve(template, clone, out);
}
```

Character Sets

Encoding iso-8859-1 in xmlparser

Question

I have some XML-Documents with `encoding="iso-8859-1"`. I am trying to parse these with `xmlparser` SAX API. In characters (`char[]`, `int`, `int`), I would like to output the content in `iso-8859-1` (Latin1) too.

With `System.out.println()` it doesn't work correctly. German umlauts result in '?' in the output stream. Internally `„, ‚, Û, >, æ` stored as `65508,65526,65532,65476,65494,65500,65503` respectively. What do I have to do to get the output in Latin1? Host system here is a SPARC Solaris 2.6.

Answer

You cannot use `System.out.println()`. You need to use an output stream which is encoding aware, for example, `OutputStreamWriter`.

You can construct an `outputstreamwriter` and use the `write(char[], int, int)` method to:

```
print.Ex:OutputStreamWriter out = new OutputStreamWriter(System.out, "8859_1");
/* Java enc string for ISO8859-1*/
```

Parsing XML Stored in NCLOB With UTF-8 Encoding

Question

I'm having trouble with parsing XML stored in NCLOB column using UTF-8 encoding. Here is what I'm running:

- Windows NT 4.0 Server
- Oracle 8i (8.1.5)
- EEJDeveloper 3.0
- JDK 1.1.8
- Oracle XML Parser v2 (2.0.2.5?)

The following XML sample that I loaded into the database contains two UTF-8 multi-byte characters:

```
<?xml version="1.0" encoding="UTF-8"?>
<G>
<A>GÂ,otingen, BrÃ ck_W</A>
</G>
```

G(0xc2, 0x82)otingen, Br(0xc3, 0xbc)ck_W

If I'm not mistaken, both multibyte characters are valid UTF-8 encodings and they are defined in ISO-8859-1 as:

```
0xC2 LATIN CAPITAL LETTER A WITH CIRCUMFLEX
0xFC LATIN SMALL LETTER U WITH DIAERESIS
```

I wrote a Java stored function that uses the default connection object to connect to the database, runs a Select query, gets the `OracleResultSet`, calls the `getCLOB` method and calls the `getAsciiStream()` method on the CLOB object. Then it executes the following piece of code to get the XML into a DOM object:

```
DOMParser parser = new DOMParser();
parser.setPreserveWhitespace(true);
parser.parse(istr);
// istr getAsciiStreamXMLDocument xmlDoc = parser.getDocument();
```

Before the stored function can do other tasks, this code throws an exception complaining that the above XML contains "Invalid UTF8 encoding".

- When I remove the first multibyte character (0xc2, 0x82) from the XML, it parses fine.
- When I do not remove this character, but connect via the JDBC Oracle: thin driver (note that now I'm not running inside the RDBMS as stored function anymore) the XML is parsed with no problem and I can do what ever I want with the XMLDocument.

I loaded the sample XML into the database using the thin JDBC driver. I tried two database configurations with WE8ISO8859P1/WE8ISO8859P1 and WE8ISO8859P1/UTF8 and both showed the same problem.

Answer

Yes, the character (0xc2, 0x82) is valid UTF-8. We suspect that the character is distorted when `getAsciiStream()` is called. Try to use `getUnicodeStream()` and `getBinaryStream()` instead of `getAsciiStream()`.

If this does not work, try print out the characters before to make sure that they are not distorted before they are sent to the parser in step: `parser.parse(istr)`

NLS support within XML

Question

I've got Japanese data stored in an `nvarchar2` field in the database. I have a dynamic SQL procedure that utilizes the PL/SQL web toolkit that allows me to access data via OAS and a browser. This procedure uses the XML Parser to correctly format the result set in XML before returning it to the browser.

My problem is that the Japanese data is returned and displayed on the browser as upside down question marks. Is there anything I can do so that this data is correctly returned and displayed as Kanji?

Answer

Unfortunately, Java and XML default character set is UTF8 while I haven't heard of any UTF8 OS nor people using it as in their database and people writing their web pages in UTF8. All this means is that you have a character code conversion problem. Answer to your last question is 'yes'. We do have both PL/SQL and Java XML parsers working in Japanese. Unfortunately, we cannot provide a simple solution that will fit in this space.

UTF-16 Encoding with XML Parser for Java V2

Question

This is my XML Document:

Documento de Prueba de gestin de contenidos. Roberto P%orez Lita

This is the way in which I parse the document:

```
DOMParser parser=new DOMParser();
parser.setPreserveWhitespace(true);
parser.setErrorStream(System.err);
parser.setValidationMode(false);
parser.showWarnings(true);
parser.parse ( new FileInputStream(new File("PruebaA3Ingles.xml")));
```

I get the following error:

```
XML-0231 : (Error) Encoding 'UTF-16' is not currently supported
```

I am using the XML Parser for Java V2_0_2_5 and I am confused because the documentation says that the UTF-16 encoding is supported in this version of the Parser. Does anybody know how can I parse documents containing spanish accents?

Answer

Oracle just uploaded a new release of V2 Parser. It should support UTF-16. Yet, other utilities still have some problems with UTF-16 encoding.

How Can I Read in Accented Characters?

Question

I need to store accented characters in my XML documents. If I manually add an accented character e.g. é, to my XML file and then attempt to parse the XML doc. with Oracle's XML Parser for Java the Parser throws the following exception:

```
'Invalid UTF-8 encoding'
```

Here's my encoding declaration in my xml header:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Aside: If I specify UTF-16 as the default encoding the Oracle XML Parser for Java states that UTF-16 is not currently supported. From within my Java program if I define a Java String object as follows:

```
String name = "éééé";
```

and programmatically generate an XML document and save it to file then the é character is correctly written out to file. Can you tell me how I can successfully read in character data consisting of accented characters. I know that I can read in accented characters once I represent them in their HEX or Decimal format within the XML document, for example:

```
&#xe9;
```

but I'd prefer not to do this.

Answer 1

You need to set the encoding based on the character set you were using when you created the xml file - I ran into this problem & solved it by setting the encoding to iso-8859-1 (western european ascii) - you may need to use something different depending on the tool and/or operating system you were using.

If you explicitly set the encoding to UTF-8 (or do not specify it at all), the parser interprets your accented character (which has an ascii value > 127) as the first byte of a UTF-8 multi-byte sequence. If the subsequent bytes do not form a valid UTF-8 sequence, you get this error.

Answer 2

This error just means that your editor is not saving the file with UTF-8 encoding. For example, it might be saving it with ISO-8859-1 encoding. Remember that the encoding is a particular scheme used to write the Unicode character number representation to disk. Just adding the string to the top of the document like:

```
<?xml version="1.0" encoding="UTF-8"?>
```

does not cause your *editor* to write out the bytes representing the file to disk using UTF-8 encoding. I believe Notepad uses UTF-8, so you might try that.

Adding XML Document as a Child

Adding an XMLDocument as a Child to Another Element

Question

I am trying to add an XMLDocument as a child to an existing element. Here's an example:

```
import org.w3c.dom.*;
import java.util.*;
import java.io.*;
import java.net.*;
import oracle.xml.parser.v2.*;
public class ggg {public static void main (String [] args) throws Exception
{
new ggg().doWork();;
public void doWork() throws Exception {XMLDocument doc1 = new XMLDocument();
Element root1=doc1.createElement("root1");
XMLDocument doc2= new XMLDocument();Element root2=doc2.createElement("root2");
root1.appendChild(root2);
doc1.print(System.out);};};
```

This reports:

```
D:\Temp\Oracle\sample>c:\jdk1.2.2\bin\javac -classpath
D:\Temp\Oracle\lib\xmlparserv2.jar;.
ggg.javaD:\Temp\Oracle\sample>c:\jdk1.2.2\bin\java -classpath
D:\Temp\Oracle\lib\xmlparserv2.jar;. gggException in thread "main"
java.lang.NullPointerException at
oracle.xml.parser.v2.XMLDOMException.(XMLDOMException.java:67) at
oracle.xml.parser.v2.XMLNode.checkDocument(XMLNode.java:919) at
```

```
oracle.xml.parser.v2.XMLNode.appendChild(XMLNode.java, Compiled Code)      at
oracle.xml.parser.v2.XMLNode.appendChild(XMLNode.java:494)                at
ggg.doWork(ggg.java:20)           at ggg.main(ggg.java:12)
```

Answer a

The following works for me:

```
DocumentFragment rootNode = new XMLDocumentFragment(); DOMParser d = new
DOMParser(); d.parse("http://.../stuff.xml");
Document doc = d.getDocument();
Element e = doc.getDocumentElement();
// Important to remove it from the first doc
// before adding it to the other doc. doc.removeChild(e);
rootNode.appendChild(e);
```

You need to use the DocumentFragment class to do this as a document cannot have more than one root.

Answer b

Actually, isn't this specifically a problem with appending a node created in another document, since all nodes contain a reference to the document they are created in? While Document Fragmentsolves this, it isn't a more than one root problem, is it? Is there a quick or easy way to convert a com.w3c.dom.Document to org.w3c.dom.DocumentFragment?

Adding an XML DocumentFragment as a Child to XMLDocument

Question

I have this piece of code:

```
XSLStyleSheet XSLProcessorStyleSheet = new XSLStyleSheet(XSLProcessorDoc,
XSLProcessorURL);
XSLStyleSheet XSLRenderersStyleSheet = new XSLStyleSheet(XSLRenderersDoc,
XSLRenderersURL);
XSLProcessor processor = new XSLProcessor();
// configure the processorprocessor.showWarnings(true);
processor.setErrorStream(System.err);
XMLDocumentFragment processedXML = processor.processXSL(XSLProcessorStyleSheet,
XMLInputDoc);
XMLDocumentFragment renderedXML = processor.processXSL(XSLRenderersStyleSheet,
processedXML);
Document resultXML = new XMLDocument();
```



```
resultXML.appendChild(renderedXML);
```

The last line causes Exception in thread "main" oracle.xml.parser.v2.

```
XMLDOMException: Node of this type cannot be added.
```

Do I have to create a root element `_every time_`, even if I know that the resulting DocumentFragment is a well formed XML Document (and of course has only one root element!)?

Answer

It happens, as you have guessed, because a Fragment can have more than one "root" element (for lack of a better term). In order to work around this, use the Node functions to extract the one root element from your fragment and cast it into an

Uninstalling Parsers

Removing XML Parser from the Database

Question

I am uninstalling a version of XML Parser and installing a newer version. How do I do that? I know that there is something like dropjava, but still there are other packages which are loaded into the schema. I want to clean out the earlier version and install the new version in a clean manner.

Answer

You'll need to write SQL to write SQL based on the USER_OBJECTS table where:

```
SELECT 'drop java class '''&#0124; &#0124;          dbms_java.longname(object_
name)&#0124; &#0124;''';
```

from user_objects where

```
OBJECT_TYPE = 'JAVA CLASS' and DBMS_JAVA.LONGNAME(OBJECT_NAME)      LIKE
'oracle/xml/parser/%'
```

This will spew out a set of DROP JAVA CLASS command which you can capture in a file using SQL*Plus': SPOOL somefilenamecommand.

Then run that spool file as a SQL script and all the right classes will be dropped.

XML Parser for Java: Installation

XMLPARSER Fails to Install

Question

I'm getting an error message when I try installing XMLPARSER:

```
loadjava -user username/manager -r -v xmlparserv2.jar
```

Error:

```
Exception in thread "main" java.lang.NoClassDefFounderr:
```

```
oracle/jdbc/driver/OracleDriver at oracle.aurora.server.tools..
```

Answer

This is a failure to find the JDBC classes111.zip in your classpath. The loadjava utility connects to the database to load your classes using the JDBC driver.

I checked 'loadjava' and the path to classes111.zip is

```
<ORACLE_HOME>/jdbc/lib/classes111.zip
```

In version 8.1.6, classes111.zip resides in:

```
<ORACLE_HOME>/jdbc/admin
```

General XML Parser Related Questions

How the XML Parser Works

Question

What does an XML Parser do?

Answer

The parser accepts any XML document giving you a tree-based API (DOM) to access or modify the document's elements and attributes as well as an event-API (SAX) that provides a listener to be registered and report specific elements or attributes and other document events.

Converting XML Files to HTML Files

Question

How do I convert XML files into HTML files?

Answer

You need to create an XSL stylesheet to render your XML into HTML. You can start with an HTML document in your desired format and populated with dummy data. Then you can replace this data with the XSLT commands that will populate the HTML with data from the XML document completing your stylesheet.

Does XML Parser Validate Against XML Schema?

Question

Does the XML Parser v2 validate against an XML Schema?

Answer

Yes. It supports both validating and non-validating modes. XML Schema is still under the development W3C XML Schema committee and is supported by Oracle. Currently, XML Parser for Java supports validating, non-validating, partial validating DTDs and XML Schemas with the modes: non-validating mode, DTD validating mode, partial validation mode, and schema validation mode.

Including Binary Data in an XML Document

Question

How do I include binary data in an XML document?

Answer

There is no way to directly include binary data within the document; however, there are two ways to work around this:

- Binary data could be referenced as an external unparsed entity that resided in a different file.
- Binary data can be uuencoded (meaning converting binary data into ASCII data) and be included in a CDATA section. The limitation on the encoding

technique is to ensure that it only produces legal characters for the CDATA section.

What is XML Schema?

Question

What is the XML Schema?

Answer

XML Schema is a W3C XML standards effort to bring the concept of data types to XML documents and in the process replace the syntax of DTDs to one based on XML. For more details, check out <http://www.w3.org/TR/xmlschema-1/> and <http://www.w3.org/TR/xmlschema-2/>. XML Schema is supported in Oracle and higher.

Oracle's Participation in Defining the XML/SQL Standard

Question

Does Oracle participate in defining the XML/XSL standard?

Answer

Oracle has representatives participating actively in the following 3C Working Groups related to XML/XSL: XML Schema, XML Query, XSL, XLink/XPointer, XML Infoset, DOM and XML Core.

XDK Version Numbers

Question

How do I determine the version number of the XDK toolkit that I downloaded?

Answer

You can find out the full version number by looking at the readme.html file included in the archive and linked off of the Release Notes page.

Inserting <, >, >= and <= in XML Documents

Question

How do I insert these characters in the XML documents: >,<,>=, and <=?

Answer

You need to use the entities < for < and > for >.

Are Namespace and Schema Supported

Question

Is support for Namespaces and Schema included?

Answer

The current XML parsers support Namespaces. Schema support is provided in Oracle9i and higher.

Using JDK 1.1.x with XML Parser for Java v2

Question

Can I use JDK 1.1.x with XML Parser v2 for Java?

Answer

v2 of XML Parser for Java has nothing to do with Java2. It is simply a designation that indicates that it is not backwards compatible with the v1 Parser and that it includes XSLT support. The v2 parser will work fine with JDK 1.1.x.

Sorting the Result on the Page

Question

I have a set of records say 100, I am showing 10 at a time, now on each column name I have made a link, on the click of the same, I want to sort the data in the page alone, based on that column. How to go about it?

Answer

It depends on how you are going about. If you are writing for IE5 alone and receiving XML data, you could just use MS's XSL to sort data in a page. If you are writing for other browser and browsers are getting data as HTML, then you have to have a sort parameter in XSQL script and use it in ORDER BY clause. Just passed it along with skip-rows parameter.

Is Oracle Needed to Run XML Parser for Java?

Question

Do I need Oracle to run the XML Parser for Java?

Answer

XML Parser for Java can be used with any of the supported version JavaVMs. The only difference is that you can load it into the database and use JServer, which is an internal JVM. For other database versions or servers, you simply run it in an external JVM and as necessary connect to a database through JDBC.

Dynamically Setting the Encoding in an XML File

Question

Is it possible to dynamically set the encodings in the XML file?

Answer

No, you need to include the proper encoding declaration in your document as per the specification. You cannot use `setEncoding()` to set the encoding for you input document. `SetEncoding()` is used with `oracle.xml.parser.v2.XMLDocument` to set the correct encoding for the printing.

Parsing a String

Question

How do I parse a string?

Answer

We do not currently have any method that can directly parse an XML document contained within a String. You would need to convert the String into an InputStream or InputSource before parsing. An easy way is to create a ByteArrayInputStream using the bytes in the String.

Displaying an XML Document**Question**

How do I display my XML document?

Answer

If you are using IE5 as your browser you can display the XML document directly. Otherwise, you can use our XSLT Processor in v2 of the parser to create the HTML document using an XSL Stylesheet. The Oracle XML Transviewer bean also allows you to view your XML document.

System.out.println() and Special Characters**Question**

I am having problems using System.out.println() with special character encoding.

Answer

You can't use System.out.println(). You need to use an output stream which is encoding aware (Ex.OutputStreamWriter). You can construct an OutputStreamWriter and use the write(char [], int, int) method to print.

```
/* Example */
OutputStreamWriter out = new OutputStreamWriter
(System.out, "8859_1");
/* Java enc string for ISO8859-1*/
```

Obtaining Ampersand from Character Data**Question**

How do I to get ampersand from character data?

Answer

You cannot have "raw" ampersands in XML data. You need to use the entity, `&`; instead. This is defined in the XML standard.

How Can We Use Special Characters in the Tags?

Question

I have a tag in XML `<COMPANYNAME>`

When we try to use "A&B", the parser gives an error with invalid character. How do we use special characters when parsing companyname tag? We are using the Oracle XML Parser for C.

Answer 1

You have to represent literal...

`&` as `&`;

`<` as `<`;

Answer 2

I think you may want to use special characters as part of XML name. For example: `<A&B>abc</A&B>`

If this is the case, using name entity doesn't solve the problem. According to XML 1.0 spec (<http://www.w3.org/TR/2000/REC-xml-20001006>), NameChar and Name are defined as follows:

[4] NameChar ::= Letter | Digit | ':' | '-' | '_' | '.' | CombiningChar | Extender

[5] Name ::= (Letter | '_' | ':') (NameChar)*

To answer your question, special character such as '&', '\$', '#', ... are not allowed to be used as NameChar. Hence, if you are creating XML document from scratch, you can use a workaround by using only valid NameChars. For example, `<A_B>`, `<AB>`, `<A_AND_B>`...

They are still readable.

If you are generating XML from external data sources such as database tables, then this is a problem. XML 1.0 does not address it.

In Oracle, the new type, XMLType, will help address this problem by offering a function which maps SQL names to XML names. This will address this problem at

the application level. The SQL to XML name mapping function will escape invalid XML NameChar in the format of `_XHHHH_` where HHHH is a Unicode value of the invalid character. For example, table name "V\$SESSION" will be mapped to XML name "V_X0024_SESSION".

At last, escaping invalid characters is a hack to give people a way to serialize names so that they can reload them somewhere else.

Parsing XML from Data of Type String

Question

How do I parse XML from data of type String?

Answer

Check out the following example:

```
/* xmlDoc is a String of xml */
byte aByteArr [] = xmlDoc.getBytes();
ByteArrayInputStream bais = new ByteArrayInputStream (aByteArr, 0,
aByteArr.length);
domParser.parse(bais);
```

Extracting Data from XML Document into a String

Question

How do I extract data from an XML document into type String?

Answer

Here is an example to do that:

```
XMLDocument Your Document;
/* Parse and Make Mods */
:
StringWriter sw = new StringWriter();
PrintWriter pw = new PrintWriter(sw);
YourDocument.print(pw);
String YourDocInString = sw.toString();
```

Disabling Output Escaping

Question

Does XML Parser for Java support Disabling Output Escaping?

Answer

Yes, since version 2.022, the parser provides an option to `xsl:text` to disable output escaping.

Using the XML Parser for Java with Oracle 8.0.5

Question

Is the XML Parser for Java only available for use with Oracle 9i? Is it possible to use with Oracle 8.0.5

Answer

The XML Parser for Java can be used with any of the supported version JavaVMs. The only difference with Oracle9i is that you can load it into the database and use JServer which is an internal VM. For 8.0.5 you simple run it externally and connect through JDBC.

Delimiting Multiple XML Documents

Question

We need to be able to read (and separate) several XML documents as a single string. One solution would be to delimit these documents using some (programmatically generated) special character that we know for sure can never occur inside an xml document. The individual documents can then be easily tokenized and extracted/parsed as required.

Has any one else done this before? Any suggestions for what character can be used as the delimiter (for instance can characters in the range `#x0-#x8` ever occur inside an xml document?)

Answer

As far as legality is concerned and you limit it to 8-bit, #x0-#x8; #xB, #xC, #xE, and #xF are not legal. HOWEVER this assumes that you preprocess the doc and not depend upon exceptions as not ALL parsers reject ALL illegal characters.

Element, which you then append to the Document.

XML and Entity-references: XML Parser for Java**Question**

1. The XML-parser for Java does not expand entity references, such as &[whatever], instead all values are null. How can I fix this?
2. It seems you cannot have international character (such as swedish characters, ...,) as values for internal entities. How does one solve this problem?

Answer

1. You probably have a simple error defining/using your entities since we've a number of regression tests that handle entity references fine. A simple example is: `<|> Alpha`, then `&status`
2. What do you set your character set encoding to be?

Can I Break up and Store an XML Document without a DDL Insert?**Question**

1. We would like to break apart an arbitrary XML document and store it in the database without creating a DDL to insert. Is this possible?
2. And as for querying, is it possible to perform hierarchical searches across XML documents?

Answer

1. No this is not possible. Either the schema must already exist or an XSL stylesheet to create the DDL from the XML must exist.
2. From Oracle8i Release 8.1.6 and higher, *interMedia Text* (now called Oracle Text) can do this.

Merging XML Documents

Question

How can I merge two XML Documents?

Answer

This is not possible with the current DOM specification. DOM2 specification may address this.

You can use a DOM-approach or an XSLT-based approach to accomplish this. If you use DOM, then you'll have to remove the node from one document before you append it into the other document to avoid ownership errors.

Here's an example of the XSL-based approach. Assume your two XML source files are:

demo1.xml

```
<messages>
  <msg>
    <key>AAA</key>
    <num>01001</num>
  </msg>
  <msg>
    <key>BBB</key>
    <num>01011</num>
  </msg>
</messages>
```

demo2.xml

```
<messages>
  <msg>
    <key>AAA</key>
    <text>This is a Message</text>
  </msg>
  <msg>
    <key>BBB</key>
    <text>This is another Message</text>
  </msg>
</messages>
```

Here is a stylesheet the "joins" demo1.xml to demo2.xml based on matching the "<key>" values.

demomerge.xsl

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output indent="yes"/>
<xsl:variable name="doc2" select="document('demo2.xml')"/>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
<xsl:template match="msg">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
    <text><xsl:value-of select="$doc2/messages/msg[key=current()/key]/text"/>
  </text>
  </xsl:copy>
</xsl:template>
</xsl:stylesheet>

```

If you use the command-line "oraxsl" to test this out, you would do:

```
$ oraxsl demo1.xml demomerge.xsl
```

And you'll get the merged result of:

```

<messages>
  <msg>
    <key>AAA</key>
    <num>01001</num>
    <text>This is a Message</text>
  </msg>
  <msg>
    <key>BBB</key>
    <num>01011</num>
    <text>This is another Message</text>
  </msg></messages>

```

Obviously not as efficient for larger-sized files as an equivalent database "join" between two tables, but this illustrates the technique if you only have XML files to work with.
Error: Cannot Find Class

Getting the Value of a Tag

Question

I am using SAX to parse an XML document. How I can get the value of a particular tag? For example, Java. How do I get the value for title? I know there are startElement, endElement, and characters methods.

Answer

During a SAX parse the *value* of an element will be the concatenation of the characters reported from after startElement to before the corresponding endElement is called.

Granting JAVASYSPRIV to User

Question

We are using Oracle XML Parser for Java on NT 4.0. When we are parsing an XML document with an external DTD we get the following error:

```
<!DOCTYPE listsamlereceipt SYSTEM
"file:/E:/ORACLE/utl_file_dir/dadm/ae.dtd">
java.lang.SecurityExceptionat
oracle.aurora.rdbms.SecurityManagerImpl.checkFile(SecurityManagerImpl.java)at
oracle.aurora.rdbms.SecurityManagerImpl.checkRead(SecurityManagerImpl.java)at
java.io.FileInputStream.<init>(FileInputStream.java)at
java.io.FileInputStream.<init>(FileInputStream.java)at
sun.net.www.MimeTable.load(MimeTable.java)at
sun.net.www.MimeTable.<init>(MimeTable.java)at
sun.net.www.MimeTable.getDefaultTable(MimeTable.java)at
sun.net.www.protocol.file.FileURLConnection.connect(FileURLConnection.java)at
sun.net.www.protocol.file.FileURLConnection.getInputStream(FileURLConnection.
java)at
java.net.URL.openStream(URL.java)at
oracle.xml.parser.v2.XMLReader.openURL(XMLReader.java:2313)at
oracle.xml.parser.v2.XMLReader.pushXMLReader(XMLReader.java:176)at
...
```

What is causing this?

Answer

Grant the JAVASYSPRIV role to your user running this code to allow it to open the external file/URL.

Including an External XML File in Another XML File: External Parsed Entities**Question**

1. I am trying to include an external XML file in another XML file. Does Oracle Parser for Java v1 and v2 support external parsed entities?
2. We are using version 1.0, because that is what is shipped to the customers with release 10.7 and 11.0 of our application. Can you refer me to this, or some other sample code to do this.

Shouldn't file b.xml be in the format:

```
<?xml version="1.0" ?>
<b>
  <ok/>
</b>
```

Does Oracle XML Parser come with a utility to parse an XML file and see the parsed output?

Answer

1. IE 5.0 will parse an XML file and show the parsed output. Just load the file like you would an HTML page.

The following works, both browsing it in IE5 as well as parsing it with Oracle XML Parser v2. Even though I'm sure it works fine in Oracle XML Parser 1.0, you should be using the latest parser version as it is faster than v1.

File: a.xml

```
<?xml version="1.0" ?>
<!DOCTYPE a [<!ENTITY b SYSTEM "b.xml">]>
<a>&b;</a>
```

File: b.xml

```
<ok/>
```

When I browse/parse a.xml I get the following:

```
<a>
  <ok/>
</a>
```

2. Not strictly. The parsed external entity only needs to be a well-formed fragment. The following program (with `xmlparser.jar` from v 1.0) in your CLASSPATH shows parsing and printing the parsed document. It's parsing here from a String but the mechanism would be no different for parsing from a file, given it's URL.

```
import oracle.xml.parser.*;
import java.io.*;
import java.net.*;
import org.w3c.dom.*;
import org.xml.sax.*;
/*
** Simple Example of Parsing an XML File from a String
** and, if successful, printing the results.
**
** Usage: java ParseXMLFromString <hello><world/></hello>
*/
public class ParseXMLFromString {
    public static void main( String[] arg ) throws IOException, SAXException {
        String theStringToParse =
            "<?xml version='1.0'?>"+
            "<hello>"+
            "  <world/>"+
            "</hello>";
        XMLDocument theXMLDoc = parseString( theStringToParse );
        // Print the document out to standard out
        theXMLDoc.print(System.out);
    }
    public static XMLDocument parseString( String xmlString ) throws
    IOException, SAXException {
        XMLDocument theXMLDoc = null;
        // Create an oracle.xml.parser.v2.DOMParser to parse the document.
        XMLParser theParser = new XMLParser();
        // Open an input stream on the string
        ByteArrayInputStream theStream =
            new ByteArrayInputStream( xmlString.getBytes() );
        // Set the parser to work in non-Validating mode
        theParser.setValidationMode(DTD_validation);
        try {
            // Parse the document from the InputStream
            theParser.parse( theStream );
        }
    }
}
```



```

        // Get the parsed XML Document from the parser
        theXMLDoc = theParser.getDocument();
    }
    catch (SAXParseException s) {
        System.out.println(xmlError(s));
        throw s;
    }
    return theXMLDoc;
}
private static String xmlError(SAXParseException s) {
    int lineNum = s.getLineNumber();
    int colNum = s.getColumnNumber();
    String file = s.getSystemId();
    String err = s.getMessage();
    return "XML parse error in file " + file +
        "\n" + "at line " + lineNum + ", character " + colNum +
        "\n" + err;
}
}

```

Where Can I Download OraXSL, The Parser's Command Line Interface?

Question

From where I can download `oracle.xml.parser.v2.OraXSL`?

Answer

It's part of our integrated XML Parser for Java V2 release. Our XML Parser, DOM, XPath implementation, and XSLT engine are nicely integrated into a single, cooperating package. http://otn.oracle.com/tech/xml/xdk_java/

Will Oracle Support Hierarchical Mapping?

Question

We are interested in using the Oracle database to primarily store XML. We would like to parse incoming XML documents and store data and tags in the database. We are concerned about the following two aspects of XML in Oracle:

Relational mapping of parsed XML data. We prefer hierarchical storage of parsed XML data. Is this a valid concern? Will XMLType in Oracle9i address this concern?

A lack of an "Ambiguous Content Mode" in the Oracle Parser for Java is limiting to our business. Are there plans to add an "Ambiguous Content Mode" to the Oracle Parser for Java?

Answer

Lots of customers initially have this concern. It depends on what kind of XML data you are storing. If you are storing XML datagrams that are really just encoding of relational information, a purchase order, for example, then you will get much better performance and much better query flexibility (via SQL) to store the data contained in the XML documents in relational tables, then on-demand reproduce an XML format when any particular data is needed to be extracted.

If you are storing documents that are more mixed-content, like legal proceeding, chapters of a book, reference manuals, and so on. Then storing them in chunks and searching them using Oracle Text's XML search capabilities is the best bet.

The book, "Building Oracle XML Applications" by Steve Muench, covers both of these storage and searching techniques with lots of examples.

See Also:

- [Chapter 8, "Searching XML Data with Oracle Text"](#)
- *Oracle9i Text Reference*

For the second point, Oracle's XML Parser implements all the XML 1.0 standard, and the XML 1.0 standard requires XML documents to have unambiguous content models, so there's no way a compliant XML 1.0 parser can implement ambiguous content models. See: <http://www.xml.com/axml/target.html#determinism>

XSLT Processor and XSL Stylesheets

HTML Error in XSL

Question

I don't know what is wrong here. This is my news_xsl.xml file:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <HTML>
```

```

<HEAD>
  <TITLE>   Sample Form   </TITLE>
</HEAD>
<BODY>
  <FORM>
    <input type="text" name="country" size="15">   </FORM>
  </BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>

```

ERROR:End tag 'FORM' does not match the start tag 'input'. Line 14, Position 12

```
</FORM>--
```

```
-----^news.xml
```

```
<?xml version="1.0" ?>
```

```
<?xml-stylesheet type="text/xsl" href="news_xsl.xsl"?>
```

```
<GREETING/>
```

Answer

Unlike in HTML, in XML you must know that every opening/starting tag should have an ending tag. So even the input that you are giving should have a matching ending tag, so you should modify your script like this:

```

<FORM>
<input type="text" name="country" size="15"> </input>
</FORM>

```

OR

```

<FORM>
<input type="text" name="country" size="15"/>
</FORM>

```

And also always remember, in XML the tags are case sensitive, unlike in HTML. So be careful.

Is <xsl:output method="html"/> Supported?

Question

Is the output method “html” supported in the recent version of the XML/XSL parser? I was trying to use the
 tag with the <xsl:output method="xml"/> declaration but I got an XSLException error message indicating a not well-formed

XML document. Then I tried the following output method declaration: `<xsl:output method="html"/>` but I got the same result.

Here's a simple XSL stylesheet I was using:

```
<?xml version="1.0"?> <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:output method="html"/>
<xsl:template match="/">      <HTML>          <HEAD></HEAD>          <BODY>
<P>          Blah blah<BR>          More blah blah<BR>          </P>
</BODY>      </HTML>      </xsl:template>
```

How do I use a not well-formed tag (like ``, `
`, etc.) in an XSL stylesheet?

Answer

We fully support all options of `<xsl:output>`. The problem here is that your XSL Stylesheet must be a well-formed XML document, so everywhere you are using the `
` element, you need to use `
` instead. `<xsl:output method="html"/>` requests that when the XSLT Engine *writes out* the result of your transformation, is a proper HTML document. What the XSLT engine reads *in* must be well-formed XML.

Question 2

Sorry for jumping in on this thread, but I have a question regarding your reply. I have an XSL stylesheet that preforms XML to HTML conversion. Everything works correctly with the exception of those HTML tags that are not well formed. Using your example if I have something like:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
.....
<input type="text" name="{NAME}" size="{DISPLAY_LENGTH}" maxLength="{LENGTH}">
</input>
.....
</xsl:stylesheet>
```

It would render HTML in the format of

```
<HTML>.....<input type="text" name="in1" size="10" maxLength="20"/>
.....
</HTML>
```

While IE can handle this Netscape can not. Is there anyway to generate completely cross browser compliant HTML with XSL?

Answer 2

If you are seeing:

```
<input ... />
```

instead of:

```
<input>
```

Then you are likely using the incorrect way of calling `XSLProcessor.processXSL()` since it appear that it's not doing the HTML output for you. Use:

```
void processXSL(style,sourceDoc,PrintWriter)
```

instead of:

```
DocumentFragment processXSL(style,sourceDoc)
```

and it will work correctly.

Netscape 4.0: Preventing XSL From Outputting <meta> Tag**Question**

I'm using `<xsl utput method="html" encoding="iso-8859-1" indent = "no" />`. Is it possible to prevent XSLT from outputting `<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">` in the head because Netscape 4.0 has difficulties with this statement. It renders the page twice.

Answer

The XSLT 1.0 Recommendation says in Section 16.2 ("HTML Output Method")...If there is a HEAD element, then the html output method should add a META element immediately after the start-tag of the HEAD element specifying the character encoding actually used.

For example:

```
<HEAD><META http-equiv="Content-Type" content="text/html; charset=EUC-JP">
```

So any XSLT 1.0-compliant engine needs to add this.

Question 2

Netscape 4.0 has following bug:

When Mozilla hits the meta-encoding tag it stops rendering the page and does a refresh. So you experience this annoying flickering. So I probably have to do a replacement in the servlets Outputstream, but I don't like doing so. Are there any alternatives.

Answer 2

Only alternatives I can think of are:

- Don't have a <HEAD> section in your HTML page. As per the XSLT specification, this will suppress the inclusion of the <META> tag.
- Don't use method="HTML" for the output. Since it defaults to "HTML" as per the specification for result trees that start with <HTML> (in any mixture of case), you'd have to explicitly set it to method="xml" or method="text".

Neither is pretty, but either one might provide a workaround.

XSL Error Messages

Question

Where can I find more info on the XSL error messages. I get the error XSL-1900, exception occurred. What does this mean? How can I find out what caused the exception?

Answer

If you are using Java, you could write Exception routines to trap errors. Using tools such as JDeveloper also helps.

The error messages of our components are usually more legible. XSL-1900 indicates possible internal error or incorrect usage.

Generating HTML: "<" Character

Question

I am trying to generate an HTML form for inputting data using column names from the user_tab_columns table and the following XSL code:

```
<xsl:template match="ROW">
<xsl:value-of select="COLUMN_NAME" />
<: lt;INPUT NAME="<xsl:value-of select="COLUMN_NAME" />>
```

```
</xsl:template>
```

although 'gt;' is generated as '>' 'lt;' is generated as '<'. How do I generate the "<" character?

Comment

Using the following:

```
<xsl:text disable-output-escaping="yes">entity-reference</xsl:text>
```

does what I need.

HTML "<" Conversion Works in oraxsl but not XSLSample.java?

Question

I can't seem to display HTML from XML. In my XML file I store the HTML snippet in an XML tag:

```
<PRE>
<body.htmlcontent>
<&#60;table width="540" border="0" cellpadding="0"
cellspacing="0">&#60;tr>&#60;td>&#60;font face="Helvetica, Arial"
size="2">&#60;!-- STILL IMAGE GOES HERE -->&#60;img
src="graphics/imagegoeshere.jpg" width="200" height="175" align="right"
vspace="0" hspace="7">&#60;!-- END STILL IMAGE TAG -->&#60;!-- CITY OR TOWN NAME
GOES FIRST FOLLOWED BY TWO LETTER STATE ABBREVIATION -->&#60;b>City, state
abbreviation&#60;/b> - &#60;!-- CITY OR TOWN NAME ENDS HERE -->&#60;!-- STORY
TEXT STARTS HERE -->Story text goes here.. &#60;!-- STORY TEXT ENDS HERE
-->&#60;/font>&#60;/td>&#60;/tr>&#60;/table>
</body.htmlcontent>
</PRE>
```

I use the following in my XSL:

```
<xsl:value-of select="body.HTMLcontent" disable-output-escaping="yes"/>
```

However, the HTML output

```
<PRE>&#60;</PRE>
```

is still outputted and all of the HTML tags are displayed in the browser. How do I display the HTML properly?

Comment

That doesn't look right. All of the < are #60; in the code with an ampersand in front of them. They are still that way when they are displayed in the browser.

Even more confusing is that it works with oraxsl, but not with XSLSample.java.

Answer

This makes sense. Here's why:

- oraxsl internally uses the: void XSLProcessor.processXSL(style,source,printwriter);
- XSLSample.java uses:DocumentFragment XSLProcessor.processXSL(style,source);

The former supports <xsl:output> and all options related to writing out output that might not be valid XML (including the disable output escaping). The latter is pure XML-to-XML tree returned, so no <xsl:output> or disabled escaping can be used since nothing's being output, just a DOM tree fragment of the result is being returned.

XSLT Examples

Question

Is there any site which has good examples or small tutorials on XSLT?

Answer

This site is an evolving tutorial on lots of different XML/XSLT/XPath-related subjects:

http://zvon.vscht.cz/ZvonHTML/Zvon/zvonTutorials_en.html

XSLT Features

Question

1. Is there a list of features of the XSLT that Oracle XDK implements?

2. So the v2 parsers implement more features of the recommendation than IE5? My first impression supports this, the use of <xsl:choose... and <xsl:if... works with the v2 parser but gives strange messages with IE5.

Answer

1. Our v2 parsers support the W3C Recommendation of w3c XSLT version 1.0 at <http://www.w3.org/TR/XSLT>.
2. You are correct. Ours is XSLT Recommendation compliant.

Using XSL To Convert XML Document To Another Form

Question

I am in the process of trying to convert an xml document from one format to another by means of an xsl (or xslt) stylesheet. Before incorporating it into my java code, I tried testing the transformation from the command line:

```
> java oracle.xml.parser.v2.oraxsl jwnemp.xml jwnemp.xsl newjwnemp.xml
```

The problem is that instead of returning the transformed xml file (newjwnemp.xml), the above command just returns a file with the xsl code from jwnemp.xsl in it. I cannot figure out why this is occurring. I have attached the two input files.

```
<?xml version="1.0"?>
<employee_data>
  <employee_row>
    <employee_number>7950</employee_number>
    <employee_name>CLINTON</employee_name>
    <employee_title>PRESIDENT</employee_title>
    <manager>1111</manager>
    <date_of_hire>20-JAN-93</date_of_hire>
    <salary>125000</salary>
    <commission>1000</commission>
    <department_number>10</department_number>
  </employee_row>
</employee_data>

<?xml version='1.0'?>
<ROWSET xmlns:xsl="HTTP://www.w3.org/1999/XSL/Transform">
  <xsl:for-each select="employee_data/employee_row">
    <ROW>
```

```
<EMPNO><xsl:value-of select="employee_number" /></EMPNO>
<ENAME><xsl:value-of select="employee_name" /></ENAME>
<JOB><xsl:value-of select="employee_title" /></JOB>
<MGR><xsl:value-of select="manager" /></MGR>
<HIREDATE><xsl:value-of select="date_of_hire" /></HIREDATE>
<SAL><xsl:value-of select="salary" /></SAL>
<COMM><xsl:value-of select="commission" /></COMM>
<DEPTNO><xsl:value-of select="department_number" /></DEPTNO>
</ROW>
</xsl:for-each>
</ROWSET>
```

Answer

This is occurring nearly 100%-likely because you have the wrong XSL namespace uri for your `xmlns:xsl="..."` namespace declaration.

If you use: `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
everything works.

If you use `xmlns:xsl="-- any other string here --"`

It will do what you're seeing.

Information on XSL?

Question

I cannot find anything about using XSL. Can you help? I would like to get an XML and XSL file to show my company what they can expect from this technology. XML alone is not very impressive for users.

Answer

A pretty good starting place for XSL is the following page:

<http://metalab.unc.edu/xml/books/bible/updates/14.html>

It shows pretty much in english what the gist of xsl is. XSL isn't really anything more than an XML file, so I don't think that it will be anymore impressive to show to a customer. There's also the main website for XSL which is:

<http://www.w3.org/style/XSL/>

XSLProcessor and Multiple Outputs?

Question

I recall seeing discussions about XSLProcessor producing more than one result from one XML and XSL. How can this can be achieved?

Answer

XML Parser 2.0.2.8 supports <ora:output> to handle this.

What Good Books for XML/XSL Can You Recommend?

Question

Can any one suggest good books for learning about XML/XSL?

Answer

There are many excellent articles, white papers, and books that describe all facets of XML technology. Many of these are available on the world wide web. The following are some of the most useful resources we have found:

- XML, Java, and the Future of the Web by Jon Bosak, Sun Microsystems
<http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm>
- XML for the Absolute Beginner by Mark Johnson, JavaWorld
http://www.javaworld.com/jw-04-1999/jw-04-xml_p.html
- XML And Databases by Ronald Bourret, Technical University of Darmstadt
<http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/>
- XMLAndDatabases.htm World Wide Web Consortium (W3C)
- XML Specifications <http://www.w3.org/XML/>
- XML.com (a broad collection of XML resources and commentary)
<http://www.xml.com/>
- Annotated XML Specification by Tim Bray, XML.com
<http://www.xml.com/axml/testxml.htm>
- The XML FAQ by the W3C XML Special Interest Group
<http://www.ucc.ie/xml/> XML.org (the industry clearing house for XML DTDs that allow companies to exchange XML data)

- <http://xml.org/>
- xDev (the DataChannel XML Developer pages) <http://xdev.datachannel.com/>

XML Developer Kits for HP/UX Platform

Question

I would like to know if there are any release plans for the XML Parser or an XDK for HP/UX platform.

Answer

HP-UX ports for our C/C++ Parser as well as our C++ Class Generator are available. Look for an announcement on <http://technet.oracle.com>

Compressing Large Volumes of XML Documents

Question

Can we compress XML documents when saving them to the database as a CLOB? If they are compressed, what is the implication of using Oracle Text (intermedia) against the documents? We have large XML documents that go up to 1 megabyte and they need to be minimized.

The main requirement is to save cost in terms of disk storage as the XML documents stored are history information (more of a datawarehouse environment). We could save a lot of disk space if we could compress the documents before storage. The searching capability is only secondary, but a big plus.

Answer a

XDK for Java support a compression mechanism in Oracle. It supports streaming compression/uncompression. The compression is achieved by removing the markup in the XML Document. The initial version does not support searching the compressed data. This is planned for a future release.

Answer b

If you want to store and search your XML docs, Oracle Text can handle this. I am sure that the size of individual document is not a problem for Oracle Text.

If you want to compress the 1megabyte docs for saving disk space/costs, Oracle Text will not be able to automatically handle a compressed XML document.

My only concern would be the performance hit to do the uncompression. If you are just worried about transmitting the XML from client to server or vice versa, then HTTP compression could be easier.

How Can I Generate an XML Document Based on Two Tables?

Question

I would like to generate an XML-document based on 2 tables with a master detail relationship between them. Suppose I have two tables :

- PARENT with columns : ID and PARENT_NAME (Key = ID)
- CHILD with columns : PARENT_ID, CHILD_ID, CHILD_NAME (Key = PARENT_ID + CHILD_ID)

And a master detail relationship between PARENT and CHILD. How can I generate a document that looks like this ?

```
<?xml version = '1.0'?>
<ROWSET>
  <ROW num="1">
    <parent_name>Bill</parent_name>
    <child_name>Child 1 of 2</child_name>
    <child_name>Child 2 of 2</child_name>
  </ROW>
  <ROW num="2">
    <parent_name>Larry</parent_name>
    <child_name>Only one child</child_name>
  </ROW>
</ROWSET>
```

Answer

You can (should) use an object view to generate an XML document from a master-detail structure. In your case:

```
create type child_type is object
(child_name <data type child_name>) ;
/
create type child_type_nst
is table of child_type ;
```

```

/

create view parent_child
as
select p.parent_name
, cast
  ( multiset
    ( select c.child_name
      from child c
      where c.parent_id = p.id
    ) as child_type_nst
  ) child_type
from parent p
/

```

A `SELECT * FROM parent_child`, processed by an SQL to XML utility would generate a valid XML document for your parent child relationship. The structure would not look like the one you have presented, though. It would be like:

```

<?xml version = '1.0'?>
<ROWSET>
  <ROW num="1">
    <PARENT_NAME>Bill</PARENT_NAME>
    <CHILD_TYPE>
      <CHILD_TYPE_ITEM>
        <CHILD_NAME>Child 1 of 2</CHILD_NAME>
      </CHILD_TYPE_ITEM>
      <CHILD_TYPE_ITEM>
        <CHILD_NAME>Child 2 of 2</CHILD_NAME>
      </CHILD_TYPE_ITEM>
    </CHILD_TYPE>
  </ROW>
  <ROW num="2">
    <PARENT_NAME>Larry</PARENT_NAME>
    <CHILD_TYPE>
      <CHILD_TYPE_ITEM>
        <CHILD_NAME>Only one child</CHILD_NAME>
      </CHILD_TYPE_ITEM>
    </CHILD_TYPE>
  </ROW>
</ROWSET>

```

Using XML Schema Processor for Java

This chapter contains the following sections:

- [Introducing XML Schema](#)
- [Oracle XML Schema Processor for Java Features](#)
- [XML Schema Processor for Java Usage](#)
- [How to Run the XML Schema for Java Sample Program](#)

Introducing XML Schema

XML Schema is being drawn up by W3C to describe the content and structure of XML documents in XML. It includes the full capabilities of DTDs so that existing DTDs can be converted to XML Schema. XML Schemas have additional capabilities over DTDs.

How DTDs and XML Schema Differ

Document Type Definition (DTD) is a mechanism provided by XML 1.0 for declaring constraints on XML markup. DTDs allow the specification of the following:

- Which elements can appear in your XML documents
- What elements can be in the elements
- The order the elements can appear

XML Schema language serves a similar purpose to DTDs (Document Type Description), but it is more flexible in specifying XML document constraints and potentially more useful for certain applications. See the following section "[DTD Limitations](#)".

Consider the XML document:

```
<?XML version="1.0">
<publisher pubid="ab1234">
  <publish-year>2000</publish-year>
  <title>The Cat in the Hat</title>
  <author>Dr. Seuss</author>
  <artist>Ms. Seuss</artist>
  <isbn>123456781111</isbn>
</publisher>
```

Consider a typical DTD for the foregoing XML document:

```
<!ELEMENT publisher (year,title, author+, artist?, isbn)>
<!ELEMENT publish-year (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
...
```


DTD Limitations

DTDs, also known as XML Markup Declarations, are considered to be deficient in handling certain applications including the following:

- Document authoring and publishing
- Exchange of metadata
- E-commerce
- Inter-database operations

DTD limitations include:

- DTD is not integrated with Namespace technology so users cannot import and reuse code
- DTD does not support data types other than character data, a limitation for describing metadata standards and database schemas
- Applications need to specify document structure constraints more flexibly than the DTD allows for

XML Schema Features

[Table 21-1](#) lists XML Schema features. Note that XML Schema features include DTD features.

Table 21–1 XML Schema Features

XML Schema Feature	DTD
Built-In Data Types <p>XML schema specifies a set of built-in datatypes. Some of them are defined by their own called primitive datatypes, and they form the basis of the type system:</p> <p>string, boolean, float, decimal, double, timeDuration, timeInstant, time, date, yearMonth, year, monthDay, day, month, binary, uriReference, ID, IDREF, ENTITY, QName.</p> <p>Others are derived datatypes that are defined in terms of primitive types.</p>	DTDs do not support data types other than character strings.
User-Defined Data Types <p>Users can derive their own datatypes from the built-in data types. There are three ways of datatype derivation: restriction, list and union. Restriction defines a more restricted data type by applying constraining facets to the base type, list simply allows a list of values of its item type, and union defines a new type whose value can be of any of its member types.</p> <p>For example, to specify that the value of publish-year type to be within a specific range:</p> <pre><SimpleType name = "publish-year"> <restriction base="year"> <minInclusive value="1970"/> <maxInclusive value="2000"/> </restriction> </SimpleType></pre> <p>The constraining facets are: length, minLength, maxLength, pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, precision, scale, encoding. Some facets only apply to certain base types.</p>	The publish-year element in the DTD example cannot be constrained further.

Table 21–1 XML Schema Features (Cont.)

XML Schema Feature	DTD
<p>Occurrence Indicators (Content Model or Structure)</p> <p>In XML Schema, the structure (called <code>complexType</code>) of the instance document or an element is defined in terms of model group and attribute group. A model group may further contain model groups or element particles, while attribute group contains attributes. Wildcards can be used in both model group and attribute group to indicate any element or attribute. There are three varieties of model group: sequence, all, and choice, representing the sequence, conjunction and disjunction relationships among particles respectively. The range of the number of occurrence of each particle can also be specified.</p> <p>Like the data type, <code>complexType</code> can be derived from other types. The derivation method can be either restriction or extension. The derived type inherits the content of the base type plus corresponding modifications. In addition to inheritance, a type definition can make references to other components. This feature allows a component being defined once and used in many other structures.</p> <p>The type declaration and definition mechanism in XML Schema is much more flexible and powerful than the DTD.</p>	<p>Control by DTDs over the number of child elements in an element are assigned with the following symbols:</p> <ul style="list-style-type: none"> ■ ? = zero or one. In the foregoing DTD example, <code>artist?</code> implied artist is optional - there may or may not be an artist. ■ * = zero or more ■ + = one or more (in the foregoing DTD example, <code>author+</code> implies more than one author is possible) ■ (none) = exactly one
<p>Identity Constraints</p> <p>XML Schema extends the concept of XML ID/IDREF mechanism with the declarations of unique, key and keyref. They are part of the type definition and allow not only attributes, but also element contents as keys. Each constraint has a scope within which it holds and the comparison is in terms of their value rather than lexical strings.</p>	
<p>Import/Export Mechanisms (Schema Import, Inclusion and Modification)</p> <p>All components of a schema need not be defined in a single schema file. XML Schema provides a mechanism of assembling multiple schemas. Import is used to integrate schemas of different namespace while inclusion is used to add components of the same namespace. Components can also be modified using redefinition when included.</p>	<p>You cannot use constructs defined in external schemas.</p>
<p>Extensibility Mechanism</p> <p>XML Schema is more flexible and supports three possible models:</p> <p>Open model - where content and attributes declared for the element are required but other content and attributes are possible</p> <p>Refinable model - content and attributes are declared for the element. Allows content and attributes declared in refined sub-types.</p> <p>Closed model - as with DTDs where additional child elements and attributes not in the element declaration are not allowed.</p>	<p>An instance of an element cannot have additional child elements and attributes not specified in the schema's element declaration.</p>

XML Schema can be used to define a class of XML documents. “Instance document” describes an XML document that conforms to a particular schema.

Although these instances and schemas need not exist specifically as “documents”, they are commonly referred to as files. They may exist as any of the following:

- Streams of bytes
- Fields in a database record
- Collections of XML Infoset “Information Items”

See Also:

- <http://www.w3.org/TR/xmlschema-0/>
- [Appendix C, "XDK for Java: Specifications and Cheat Sheets"](#)
- *Oracle9i XML Reference*

Oracle XML Schema Processor for Java Features

Oracle XML Schema Processor for Java has the following features:

- Supports simple and complex types
- Built on the Oracle XML Parser for Java v2
- Supports the following W3C XML Schema Working Drafts
 - XML Schema Part 0: Primer
 - XML Schema Part 1: Structures
 - XML Schema Part 2: Datatypes

Supported Character Sets

XML Schema Processor for Java supports documents in the following encodings:

- BIG
- EBCDIC-CP-*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP

- ISO-2022-KR
- ISO-8859-1to -9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift_JIS
- US-ASCII
- UTF-8
- UTF-16

What's Needed to Run XML Schema Processor for Java

To run XML Schema Processor for Java, you need the following:

- Operating Systems: Any OS with Java 1.1.x support
- JAVA: JDK 1.1.x. or above.

Online Documentation

Documentation for Oracle XML Schema Processor for Java is located in the doc/ directory in your install area.

Release Specific Notes

The readme.html file in the root directory of the archive, contains release specific information including bug fixes, API additions,...

Oracle XML Schema Processor is an early adopter release and is written in Java. It includes the production release of the XML Parser for Java v2.

Standards Conformance

The XML Schema Processor conforms to the following W3C standards:

- XML Schema Part 0: Primer
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes

XML Schema Processor for Java Directory Structure

[Table 21–2](#) lists the directory structure after installing XML Schema Processor for Java on Windows NT. Installation on UNIX renders the same structure.

Table 21–2 *Directory Structure for a Windows NT Installation of XML Schema Processor*

Directory and File	Description
license.html	copy of license agreement
readme.html	release and installation notes
doc\	directory for documents
lib\	directory for class files
sample\	sample code files

XML Schema Processor for Java Usage

As shown in [Figure 21–1](#), Oracle's XML Schema processor performs two major tasks:

- A builder assembles schema from schema XML documents
- A validator use the schema to validate instance document.

When building the schema, the builder first calls the DOM Parser to parse the schema XML documents into corresponding DOM trees. It then compiles them into an internal schema object. The validator works as a filter between the SAX parser and your applications for the instance document. The validator takes SAX events of the instance document as input and validates them against the schema. If the validator detects any invalid XML component it sends an error message. The output of the validator is:

- Input SAX events
- Default values it supplies
- Post-Schema Validation (PSV) information

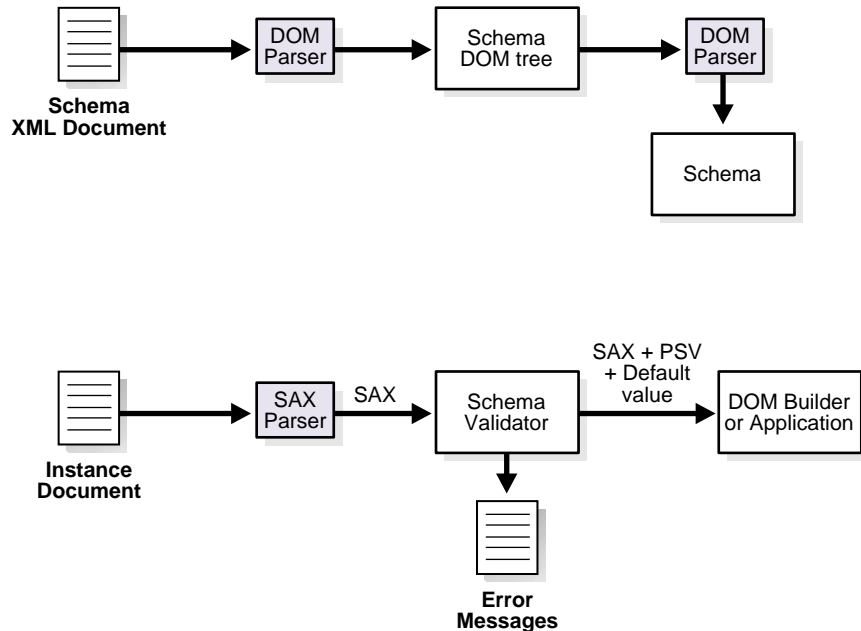
The API of the XML Schema Processor for Java is simple. You can either use either of the following:

- `setSchemaValidationMode()` in the `DOMParser` as shown in "[XML Schema for Java Example 7: XSDSample.java](#)"

- Explicitly build the schema using `XSDBuilder` and set the schema for `XMLParser` as shown in "XML Schema for Java Example 8: [XSDataSetSchema.java](#)".

There is no clean up call similar to `xmlclean`. If you need to release all memory and reset the state before validating a new XML document, terminate the context and start over.

Figure 21-1 XML Schema Processor for Java Usage



See Also: *Oracle9i XML Reference*, under XDK for Java, XML Schema Processor

How to Run the XML Schema for Java Sample Program

XML Schema Processor for Java sample/ directory contains sample XML applications that illustrate how to use Oracle XML parser with XML Schema Processor for Java. The sample Java file provided in this directory is `XSDSample`, a sample driver that processes XML instance documents. To run the sample program, carry out the following:

1. Execute "**make**" to generate .class files.
2. Add `xmlparserv2.jar`, `xschema.jar`, and the current directory to the `CLASSPATH`.
3. Run the sample program with the **report.xml** file, as follows:

```
java XSDSample report.xml
java XSDSetSchema report.xsd report.xml
```

XML Schema Processor uses the XMLSchema specification from "report.xsd" to validate the contents of "report.xml"

4. Run the sample program with the **catalogue.xml** file, as follows:

```
java XSDSample catalogue.xml
java XSDSetSchema cat.xsd catalogue.xml
```

XML Schema Processor uses the XMLSchema specification from "cat.xsd" to validate the contents of "catalogue.xml"

5. The following are examples with XMLSchema errors:

```
java XSDSample catalogue_e.xml
java XSDSample report_e.xml
```

XML Schema Processor generates error messages.

MakeFile

```
# Makefile for sample java files
# =====

.SUFFIXES : .java .class

CLASSES = XSDSample.class

# Change it to the appropriate separator based on the OS.
PATHSEP = :

# XML Parser V2 jar file
XMLPARSER = ../lib/xmlparserv2.jar

# XMLSchema jar file
XSHEMA = ../lib/xschema.jar

# Assumes that the CLASSPATH contains JDK classes.
```



```

CLASSPATH := $(CLASSPATH)$(PATHSEP)$(XMLPARSER)$(PATHSEP)$(XSHEMA)

%.class: %.java
/usr/local/packages/jdk1.2/bin/javac -classpath "$(CLASSPATH)" $<

# make all class files
all: $(CLASSES)

```

XML Schema for Java Example 1: cat.xsd

This is the sample XML Schema Definition file that inputs XSDSetSchema.java program. XML Schema Processor uses the XMLSchema specification from cat.xsd to validate the contents of catalogue.xml.

```

<?xml version="1.0"?>
schema xmlns="http://www.w3.org/1999/XMLSchema"
        targetNamespace="http://www.somewhere.org/BookCatalogue"
        xmlns:catd = "http://www.somewhere.org/Digest"
        xmlns:cat = "http://www.somewhere.org/BookCatalogue">

<import namespace = "http://www.somewhere.org/Digest"
        schemaLocation = "catd.xsd" />

<element name="BookCatalogue">
  <complexType>
    <all>
      <element ref="cat:Book" minOccurs="0" maxOccurs="*" />
      <element name="Digest" type="catd:Digest" minOccurs="0" maxOccurs="*" />
    </all>
  </complexType>
</element>
<element name="Book">
  <complexType content="mixed">
    <group ref="cat:Book" />
    <attribute name="number" type="integer" />
    <attribute name="volumeName" type="string" />
    <attribute name="volumeNumber" type="integer" />
  </complexType>
</element>
<group name="Book">
  <all>
    <element ref="cat:Title" />
    <element ref="cat:Author" minOccurs="0" maxOccurs="1" />
    <element ref="cat:Date" />
    <element ref="cat:ISBN" />
  </all>
</group>

```

```
        <element ref="cat:Publisher"/>
    </all>
</group>
<element name="Title" type="string"/>
<element name="Author" type="string"/>
<element name="Date" type="date"/>
<element name="ISBN" type="string"/>
<element name="Publisher" type="string"/>
</schema>
```

XML Schema for Java Example 2: catalogue.xml

This is the sample XML file that is validated by XML Schema processor against the XML Schema Definition file, cat.xsd, using the program, XSDSetSchema.java.

```
<?xml version="1.0"?>
<BookCatalogue xmlns =
    "http://www.somewhere.org/BookCatalogue"
    xmlns:xsi =
    "http://www.w3.org/1999/XMLSchema/instance"
    xsi:schemaLocation =
    "http://www.somewhere.org/BookCatalogue
    cat.xsd">

    <Book number="11" volumeName="any" volumeNumber="1">
        <Date>July, 1998</Date>
        <Title>My Life and Times</Title>
        <Author>Paul McCartney</Author>
        <ISBN>1111-12021-43892</ISBN>
        <Publisher>McMillin Publishing</Publisher>
    </Book>
    <Digest>
        <Title>Book Revview</Title>
        <Volume>42</Volume>
        <Publisher>McMillin Publishing</Publisher>
    </Digest>
</BookCatalogue>
```

XML Schema for Java Example 3: catalogue_e.xml

When XML Schema Processor processes this sample XML file using XSDSample.java, it generates XML Schema errors.

```
<?xml version="1.0"?>
<BookCatalogue xmlns =
```

```

        "http://www.somewhere.org/BookCatalogue"
xmlns:xsi =
    "http://www.w3.org/1999/XMLSchema/instance"
xsi:schemaLocation =
    "http://www.somewhere.org/BookCatalogue
    cat.xsd">

<Book number="11k" volumeName="any" volumeNumber="1">
    <Date>July, 1998</Date>
    <Title>My Life and Times</Title>
    <Author>Paul McCartney</Author>
    <ISBN>1111-12021-43892</ISBN>
    <Publisher>McMillin Publishing</Publisher>
</Book>
<Digest>
    <Title>Book Revwiew</Title>
    <Volume>42</Volume>
    <Author>McMillin Publishing</Author>
</Digest>
</BookCatalogue>

```

XML Schema for Java Example 4: report.xml

This is the sample XML file that is validated by XML Schema processor against the XML Schema Definition file, report.xsd, using the program, XSDSetSchema.java.

```

<purchaseReport
    xmlns='http://www.example.com/Report '
    period="P3M" periodEnding="1999-12-31"
    xmlns:xsi = "http://www.w3.org/1999/XMLSchema/instance"
    xsi:schemaLocation="http://www.example.com/Report report.xsd">

    <regions>
        <zip code="95819">
            <part number="872-AA" quantity="1"/>
            <part number="926-AA" quantity="1"/>
            <part number="833-AA" quantity="1"/>
            <part number="455-BX" quantity="1"/>
        </zip>
        <zip code="63143">
            <part number="755-KY" quantity="4"/>
        </zip>
    </regions>

    <parts>

```

```
<partSpec number="872-AA">Lawnmower</partSpec>
<partSpec number="926-AA">Baby Monitor</partSpec>
<partSpec number="833-AA">Lapis Necklace</partSpec>
<partSpec number="455-BX">Sturdy Shelves</partSpec>
<partSpec number="755-KY">Sturdy Shelves</partSpec>
</parts>

</purchaseReport>
```

XML Schema for Java Example 5: report.xsd

This is the sample XML Schema Definition file that inputs XSDSetSchema.java program. XML Schema Processor uses the XMLSchema specification from report.xsd to validate the contents of report.xml.

```
<schema targetNamespace='http://www.example.com/Report'
  xmlns='http://www.w3.org/1999/XMLSchema'
  xmlns:r='http://www.example.com/Report'
  xmlns:xipo='http://www.example.com/IPO'
  elementFormDefault="qualified">

  <element name="purchaseReport">
    <complexType>
      <element name="regions" type="r:RegionsType"/>
      <element name="parts" type="r:PartsType"/>
      <attribute name="period" type="timeDuration"/>
      <attribute name="periodEnding" type="date"/>
    </complexType>
    <unique name="pZipCode">
      <selector>regions/zip</selector>
      <field>@code</field>
    </unique>
    <key name="pNumKey">
      <selector>parts/part</selector>
      <field>@number</field>
    </key>
    <keyref name="pKeyRef" refer="pNumKey">
      <selector>regions/zip/part</selector>
      <field>@number</field>
    </keyref>
  </element>
  <complexType name="RegionsType">
    <element name="zip" minOccurs="1" maxOccurs="unbounded">
      <complexType>
        <element name="part" maxOccurs="unbounded">
```

```

    <complexType content="empty">
      <attribute name="number" type="r:Sku"/>
      <attribute name="quantity" type="positiveInteger"/>
    </complexType>
  </element>
  <attribute name="code" type="positiveInteger"/>
</complexType>
</element>
</complexType>

<complexType name="PartsType">
  <element name="partSpec" minOccurs="1" maxOccurs="unbounded">
    <complexType content="textOnly">
      <attribute name="number" type="r:Sku"/>
    </complexType>
  </element>
</complexType>
<simpleType name="Sku" base="string">
  <pattern value="\d{3}-[A-Z]{2}"/>
</simpleType>
</schema>

```

XML Schema for Java Example 6: report_e.xml

When XML Schema Processor processes this sample XML file using XSDSample.java, it generates XML Schema errors.

```

<purchaseReport
  xmlns='http://www.example.com/Report'
  period="P3M" periodEnding="1999-12-31"
  xmlns:xsi = "http://www.w3.org/1999/XMLSchema/instance"
  xsi:schemaLocation="http://www.example.com/Report report.xsd">
  <regions>
    <zip code="95819">
      <part number="872-AA" quantity="1"/>
      <part number="926-AA" quantity="1"/>
      <part number="833-AAA" quantity="1"/>
      <part number="455-BX" quantity="1"/>
    </zip>
    <zip code="63143">
      <part number="755-KY" quantity="4"/>
    </zip>
  </regions>
  <parts>
    <partSpec number="872-AA">Lawnmower</partSpec>

```

```
<partSpec number="926-AA">Baby Monitor</partSpec>
<partSpec number="833-AA">Lapis Necklace</partSpec>
<partSpec number="455-BX">Sturdy Shelves</partSpec>
<partSpec number="755-KY">Sturdy Shelves</partSpec>
</parts>
</purchaseReport>
```

XML Schema for Java Example 7: XSDSample.java

```
//import oracle.xml.parser.schema.*;
import oracle.xml.parser.v2.*;
import java.net.*;
import java.io.*;
import org.w3c.dom.*;
import java.util.*;

public class XSDSample
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 1)
        {
            System.out.println("Usage: java XSDSample <filename>");
            return;
        }
        process (args[0]);
    }

    public static void process (String xmlURI) throws Exception
    {
        DOMParser dp = new DOMParser();
        URL url = createURL (xmlURI);

        // Set Schema Validation to true
        dp.setSchemaValidationMode(true);
        dp.setValidationMode(false);
        dp.setPreserveWhitespace (true);

        dp.setErrorStream (System.out);

        try
        {
            System.out.println("Parsing "+xmlURI);
            dp.parse (url);
        }
    }
}
```

```
        System.out.println("The input file <"+xmlURI+"> parsed without
errors");
    }
    catch (XMLParseException pe)
    {
        System.out.println("Parser Exception: " + pe.getMessage());
    }
    catch (Exception e)
    {
        System.out.println("NonParserException: " + e.getMessage());
    }
}

// Helper method to create a URL from a file name
static URL createURL(String fileName)
{
    URL url = null;
    try
    {
        url = new URL(fileName);
    }
    catch (MalformedURLException ex)
    {
        File f = new File(fileName);
        try
        {
            String path = f.getAbsolutePath();
            // This is a bunch of weird code that is required to
            // make a valid URL on the Windows platform, due
            // to inconsistencies in what getAbsolutePath returns.
            String fs = System.getProperty("file.separator");
            if (fs.length() == 1)
            {
                char sep = fs.charAt(0);
                if (sep != '/')
                    path = path.replace(sep, '/');
                if (path.charAt(0) != '/')
                    path = '/' + path;
            }
            path = "file://" + path;
            url = new URL(path);
        }
        catch (MalformedURLException e)
        {

```

```
        System.out.println("Cannot create url for: " + fileName);
        System.exit(0);
    }
}
return url;
}
}
```

XML Schema for Java Example 8: XSDSetSchema.java

When this example is run with `cat.xsd` and `catalogue.xml`, XML Schema Processor uses the XMLSchema specification from `cat.xsd` to validate the contents of `catalogue.xml`.

When this example is run with `report.xsd` and `report.xml`, XML Schema Processor uses the XMLSchema specification from `cat.xsd` to validate the contents of `report.xml`.

```
import oracle.xml.parser.schema.*;
import oracle.xml.parser.v2.*;

import java.net.*;
import java.io.*;
import org.w3c.dom.*;
import java.util.*;

public class XSDSetSchema
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 2)
        {
            System.out.println("Usage: java XSDSample <schema_file> <xml_file>");
            return;
        }

        XSDBuilder builder = new XSDBuilder();
        URL url = createURL(args[0]);

        // Build XML Schema Object
        XMLSchema schemadoc = (XMLSchema)builder.build(url);
        process(args[1], schemadoc);
    }
}
```



```
public static void process(String xmlURI, XMLSchema schemadoc)
throws Exception
{

    DOMParser dp = new DOMParser();
    URL url = createURL (xmlURI);

    // Set Schema Object for Validation
    dp.setXMLSchema(schemadoc);
    dp.setValidationMode(XMLParser.SCHEMA_VALIDATION);
    dp.setPreserveWhitespace (true);

    dp.setErrorStream (System.out);

    try
    {
        System.out.println("Parsing "+xmlURI);
        dp.parse (url);
        System.out.println("The input file <"+xmlURI+"> parsed without
errors");
    }
    catch (XMLParseException pe)
    {
        System.out.println("Parser Exception: " + pe.getMessage());
    }
    catch (Exception e)
    {
        System.out.println ("NonParserException: " + e.getMessage());
    }
}

// Helper method to create a URL from a file name
static URL createURL(String fileName)
{
    URL url = null;
    try
    {
        url = new URL(fileName);
    }
    catch (MalformedURLException ex)
    {
        File f = new File(fileName);
        try
        {
```

```
String path = f.getAbsolutePath();
// This is a bunch of weird code that is required to
// make a valid URL on the Windows platform, due
// to inconsistencies in what getAbsolutePath returns.
String fs = System.getProperty("file.separator");
if (fs.length() == 1)
{
    char sep = fs.charAt(0);
    if (sep != '/')
        path = path.replace(sep, '/');
    if (path.charAt(0) != '/')
        path = '/' + path;
}
path = "file://" + path;
url = new URL(path);
}
catch (MalformedURLException e)
{
    System.out.println("Cannot create url for: " + fileName);
    System.exit(0);
}
}
return url;
}
}
```

XML Class Generator for Java

This chapter contains the following sections:

- [Accessing XML Class Generator for Java](#)
- [XML Class Generator for Java: Overview](#)
- [Oracg Command Line Utility](#)
- [Class Generator for Java: XML Schema](#)
- [Using XML Class Generator for Java with XML Schema](#)
- [Using XML Class Generator for Java with DTDs](#)
- [Examples Using XML Java Class Generator with DTDs and XML Schema](#)
 - [XML Class Generator for Java, DTD Example 1a: Application — SampleMain.java](#)
 - [XML Class Generator for Java, DTD Example 1d: TestWidl.java](#)
 - [XML Class Generator for Java, Schema Example 1b: Application, CarDealer.java](#)
 - [XML Class Generator for Java, Schema Example 2b: BookCatalogue.java](#)
 - [XML Class Generator for Java, Schema Example 3b: Application — TestPo.java](#)
- [Frequently Asked Questions \(FAQs\): Class Generator for Java](#)

Accessing XML Class Generator for Java

The Oracle XML Class Generator for Java is provided with Oracle's XDK for Java. It is located at `$ORACLE_HOME/xdk/java/classgen`. It is also available for download from the OTN site: <http://otn.oracle.com/tech/xml>.

XML Class Generator for Java: Overview

XML Class Generator for Java creates Java source files from an XML DTD or XML Schema Definition. This is useful in the following situations:

- When an application wants to send an XML message to another application based on agreed-upon DTDs or XML Schemas.
- As the back end of a web form to construct an XML document.

The generated classes can be used to programmatically construct XML documents. XML Class Generator for Java also optionally generates javadoc comments on the generated source files. XML Class Generator for Java requires the XML Parser for Java and the XML Schema Processor for Java. It works in conjunction with XML Parser for Java, which parses the DTD (or XML Schema) and sends the parsed XML document to the Class Generator.

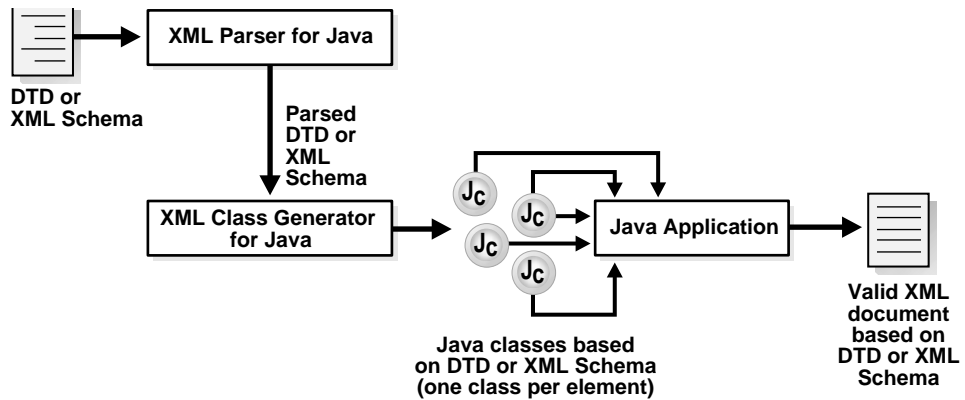
XML Class Generator for Java consists of the following two class generators:

- ***DTD Class Generator***
- ***XML Schema Class Generator***

These can both be invoked from command line utility, `oracg`.

[Figure 22-1](#) provides an overview of how XML Class Generator for Java is used.

Figure 22–1 XML Class Generator for Java: Overview



Note: The clause, “one class per element” does not apply to the XML Schema Class Generator for Java.

See Also: [Figure 3–7, "Generating XML Documents Using XDK for Java" in Chapter 3, "Oracle XML Developer Kits \(XDKs\) and Components: Overview and General FAQs"](#).

Oracg Command Line Utility

The `oracg` command line utility is used to invoke the DTD or Schema Class Generator for Java, depending on the input arguments. [Table 22–1](#) lists the `oracg` arguments.

Table 22–1 Class Generator for Java: `oracg` Command Line Arguments

<code>oracg</code> Arguments	Description
- h	Prints the help message text
- d <dtd file>	DTD file (.dtd file)
- s <schema file>	Schema file (.xsd file)
- o <Output dirname>	Specifies the output directory
- c	Comment option

Table 22–1 Class Generator for Java: oracg Command Line Arguments

oracg Arguments	Description
- p <package name/s>	Specifies the package names corresponding to namespace

Class Generator for Java: XML Schema

XML Class Generator for Java's *XML Schema* Class Generator has the following features:

- It generates a Java class for each top level element, that is, global elements `simpleType` element and `complexType` element.
- Classes corresponding to the top level elements, that is, global elements, extend the `CGXSDElement`.
- The type hierarchy among the elements is maintained in the generated Java classes. If the `complexType` or `simpleType` element extends any other `complexType` or `simpleType` element, then the class corresponding to them extends the base type `simpleType` or `complexType` element. Otherwise, they extend the `CGSXDElement` class.

Namespace Features

XML Schema Class Generator also supports the following namespace features:

- **Package Name Creation.** For each namespace, a package is created and corresponds to the elements in the namespace — the Java classes are generated in *that* package.
 - If there is no namespace defined, then the classes are generated in the default package.
 - If `targetNamespace` is specified in the schema, then a package name is required to generate the classes.
- If there is a namespace defined then the user needs to specify the package name through the command line utility. The number of packages specified should match the command line arguments corresponding to the package names.
- **Symbol Spaces.** A single distinct symbol space is used within a given target namespace for each kind of definition and declaration component identified in XML Schema. The exceptions for this is when symbol space is shared between simple type and complex type.

In a given symbol space, names are unique, but the same name may appear in more than one symbol space without conflict. For example, the same name can appear in both a type definition and an element declaration, without conflict or necessary relation between the two. In such conflict situation, the class name generated corresponding to the `simpleType/complexType` element is appended with the key word "Type".

- To avoid conflict, any methods which take the 'type' of an element (corresponding to which there is a generated Java class) as parameter, take the fully resolved name with the package name.

Using XML Class Generator for Java with XML Schema

[Figure 22-2](#) shows the calling sequence used when generating classes with XML Class Generator for Java with XML Schema.

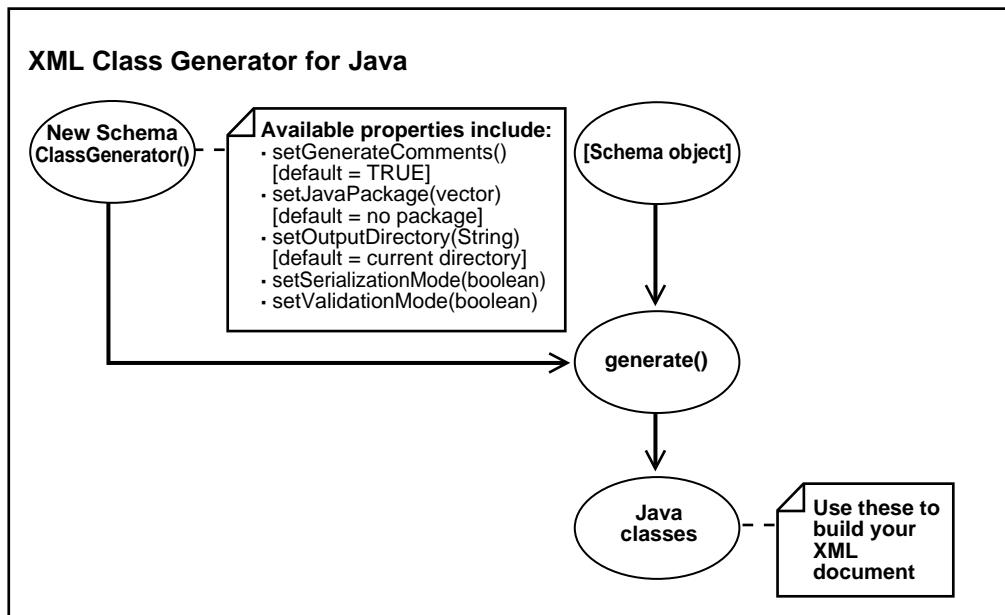
XML Java Class Generator with XML Schema operates as follows:

1. A new `SchemaClassGenerator()` class is initiated and inputs the `generate()` method. The available properties for class, `SchemaClassGenerator()` include:
 - `setGeneraterComments()`, with default = TRUE
 - `setJavaPackage(string)`, with default = no package
 - `setOutputDirectory(string)`, with default = current directory
2. If an XML Schema is used, the Schema object returned using `getDocType()` from the `parseSchema()` method, is also input. See also [Figure 20-4, "XML Parser for Java: DOMParser\(\)"](#) on page 20-18 from [Chapter 20, "Using XML Parser for Java"](#).
3. The `generate()` method generates Java classes which can then be used to build your XML document.

To generate classes using XML Class Generator for Java with XML Schema, follow the guidelines described in the following sections:

- [Generating Top Level Element Classes](#) on page 22-6
- [Generating Top Level ComplexType Element Classes](#) on page 22-7
- [Generating SimpleType Element Classes](#) on page 22-7

Figure 22–2 Generating Classes Using Class Generator for Java with XML Schema



Generating Top Level Element Classes

The following lists guidelines for using XML Schema Class Generator for Java when generating top level element classes:

- A class corresponding to the element name is generated in the package associated with the namespace.
- The element has a method called `setType` to set the type of the element in the element class. The `setType` takes fully resolved package name to avoid conflict.
- If the element has an inline `simpleType` or `complexType`, a public static class inside the element class is created which follows all the rules specified in the `simpleType/complexType`. The name of the public static class, is the element name suffixed by `Type`. For example, if the element name is `PurchaseOrder` and `PurchaseOrder` has an inline `complexType` definition, then the public static inner class will have the name `PurchaseOrder_Type`

- The name clash in class names between elements and complexType using “Type” as suffix.
- The element name and namespace is stored inside the element class (which could be used for serialization and validation)
- A validate method is provided inside the elements to accept an XMLSchema object to validate.
- A print method is provided inside the element to print the node.

Generating Top Level ComplexType Element Classes

The following lists guidelines for using XML Schema Class Generator for Java when generating top level complexType element classes:

- If the complexType element is a top level element, then a class is generated in the package associated with the namespace. If the complexType element extends a base type element, then the class corresponding to the complexType element also extends the base Type element. Otherwise, it extends the CGXSDElement class.
- The class contains fields corresponding to the attributes. The fields are made protected, so that they can be accessed from subtypes. The fields are added only for the attributes that not present in the base type.
- The class contains methods to set and get attributes.
- For each local element, a public static class is created exactly similar to top level elements, except that it will be completely inside the complexType class.

Generating SimpleType Element Classes

The following lists guidelines for using XML Schema Class Generator for Java when generating top level simpleType element classes:

- A class is generated for each top level simpleType element
- The hierarchy of the simpleType element is maintained in the generated class. If the simpleType element extends a base class then the class corresponding to the simpleType element also extends the base class corresponding to the base element. Otherwise the simpleType element extends the CGXSDElement class.
- If the simpleType element extends the schema data type, then the class extends the class corresponding to the schema data type. For example, if the

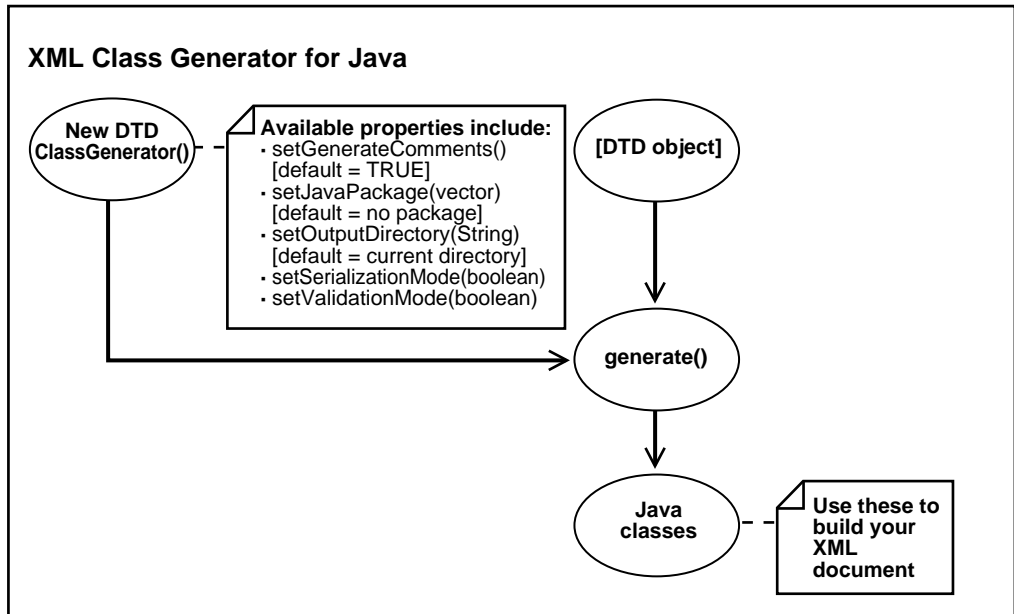
base type is a string, then the schema equivalent class is taken as `XSDStringType`, and so on.

- The class contains a field to store the `simpleType` value.
- The constructor of the `simpleType` element class sets the schema facets.
- The constructor sets the `simpleType` data value (`XSDDataValue`) in the constructor after validating against the facets.

Using XML Class Generator for Java with DTDs

Figure 22-3 shows the calling sequence of XML Java Class Generator with DTDs:

1. A new `DTDClassGenerator()` class is initiated and inputs the `generate()` method. Available properties for class, `DTDClassGenerator()` are:
 - `setGeneraterComments()`, with default = TRUE
 - `setJavaPackage(string)`, with default = no package
 - `setOutputDirectory(string)`, with default = current directory
2. If a DTD is used, the DTD object returned using `getDocType()` from the `parseDTD()` method, is also input. See also [Figure 20-4, "XML Parser for Java: DOMParser0"](#) on page 20-18 from [Chapter 20, "Using XML Parser for Java"](#).
3. The `generate()` method generates Java classes which can then be used to build your XML document.

Figure 22–3 *Generating Classes Using XML Class Generator for Java and DTDs***See Also:**

- [Chapter 3, "Oracle XML Developer Kits \(XDKs\) and Components: Overview and General FAQs", "Using Oracle XML Components to Generate XML Documents: Java" on page 3-17](#)
- [Appendix C, "XDK for Java: Specifications and Cheat Sheets"](#)
- *Oracle9i XML Reference*

Examples Using XML Java Class Generator with DTDs and XML Schema

[Table 22–2](#) lists the example files and directories supplied in `$ORACLE_HOME`:

Table 22–2 XML Class Generator for Java Example Files

Example File	Description
Makefile	Makefile used to compile and run the demo in Unix.
Make.bat	Makefile used to compile and run the demo in Windows
SampleMain.java	Sample application to generate Java source files based on a DTD.
Widl.dtd	Sample DTD.
Widl.xml	Sample XML file based on Widl.dtd.
TestWidl.java	Sample application to construct an XML document using the Java source files generated by SampleMain.
car.xsd	Sample XML Schema
CarDealer.java	Sample application to construct an XML document using the java source generated from car.xsd.
book.xsd	Sample XML Schema
BookCatalogue.java	Sample application to construct an XML document using the Java sources generated from book.xsd
po.xsd	Sample XML Schema
TestPo.java	Sample application to construct an XML document using the Java sources generated from po.xsd.

Running XML Class Generator for Java — DTD Examples

To run the XML Class Generator for Java DTD sample programs, use;

```
make target 'dtd'
```

then follow these steps:

1. Compile and run `SampleMain` to generate the Java source files, using the commands:

```
javac SampleMain.java
java SampleMain -root WIDL Widl.dtd
```

or

```
java SampleMain Widl.xml
```

2. Set the CLASSPATH to contain 'classgen.jar', 'xmlparser.jar', and the current directory.
3. Compile the Java source files generated by SampleMain, that is., BINDING.java, CONDITION.java, REGION.java, SERVICE.java, VARIABLE.java, and WIDL.java, using the command:

```
javac *.java
```

4. Run the test application to print the XML Document using the commands:

```
javac TestWidl.java  
java TestWidl
```

The output is stored in Widl_out.txt

Running XML Class Generator for Java — XML Schema Examples

To run the XML Class Generator for Java Schema sample programs, use:

```
make target 'schema'
```

There are three Schema samples: car.xsd, book.xsd, po.xsd

The classes are generated using `oracg` utility. For example, the classes corresponding to car.xsd can be generated from the command line:

```
oracg -c -s car.xsd -p package1
```

The classes are generated in the directory, package1.

When `Makefile` is used to run the schema class generator demo:

- Classes corresponding to car.xsd are generated in directory package1. Demo program, CarDealer.java, tests the generated classes. The output of CarDealer.java is stored in file, car_out.txt.
- Classes corresponding to book.xsd are generated in directory package2. Demo program BookCatalogue.java tests the generated classes. The output is stored in the file, book_out.txt.

- Classes corresponding to po.xsd are generated in directory package3. Demo program TestPo.java tests the generated classes. The output is stored in the file po_out.txt

The following Class Generator using DTD examples are included here:

- [XML Class Generator for Java, DTD Example 1a: Application — SampleMain.java](#)
- [XML Class Generator for Java, DTD Example 1b: DTD Input — widl.dtd](#)
- [XML Class Generator for Java, DTD Example 1c: Input — widl.xml](#)
- [XML Class Generator for Java, DTD Example 1d: TestWidl.java](#)
- [XML Class Generator for Java, DTD Example 1e: XML Output — widl.out](#)

XML Class Generator for Java, DTD Example 1a: Application — SampleMain.java

```
/**
 * This program generates the classes for a given DTD using
 * XML DTD Class Generator. A DTD file or an XML document which is
 * DTD compliant is given as input parameters to this application.
 */

import java.io.File;
import java.net.URL;
import oracle.xml.parser.v2.DOMParser;
import oracle.xml.parser.v2.DTD;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.classgen.DTDClassGenerator;

public class SampleMain
{

    public SampleMain()
    {
    }

    public static void main (String args[])
    {
        // Validate the input arguments
        if (args.length < 1)
        {
            System.out.println("Usage: java SampleMain "+
                               "[-root <rootName>] <fileName>");
            System.out.println("fileName\t Input file, XML document or " +
```

```
        "external DTD file");
    System.out.println("-root <rootName> Name of the root Element " +
        "(required if the input file is an external DTD)");
    return ;
}

// ty to open the XML Document or the External DTD File
try
{
    // Instantiate the parser
    DOMParser parser = new DOMParser();
    XMLDocument doc = null;
    DTD dtd = null;

    if (args.length == 3)
    {
        parser.parseDTD(fileToURL(args[2]), args[1]);
        dtd = (DTD)parser.getDoctype();
    }
    else
    {
        parser.setValidationMode(true);
        parser.parse(fileToURL(args[0]));
        doc = parser.getDocument();
        dtd = (DTD)doc.getDoctype();
    }

    String doctype_name = null;

    if (args.length == 3)
    {
        doctype_name = args[1];
    }
    else
    {
        // get the Root Element name from the XMLDocument
        doctype_name = doc.getDocumentElement().getTagName();
    }

    // generate the Java files...
    DTDClassGenerator generator = new DTDClassGenerator();

    // set generate comments to true
    generator.setGenerateComments(true);
}
```

```

        // set output directory
        generator.setOutputDirectory(".");

        // set validating mode to true
        generator.setValidationMode(true);

        // generate java src
        generator.generate(dtd, doctype_name);

    }
    catch (Exception e)
    {
        System.out.println ("XML Class Generator: Error " + e.toString());
        e.printStackTrace();
    }
}

static public URL fileToURL(String sfile)
{
    File file = new File(sfile);
    String path = file.getAbsolutePath();
    String fSep = System.getProperty("file.separator");
    if (fSep != null && fSep.length() == 1)
        path = path.replace(fSep.charAt(0), '/');
    if (path.length() > 0 && path.charAt(0) != '/')
        path = '/' + path;
    try
    {
        return new URL("file", null, path);
    }
    catch (java.net.MalformedURLException e)
    {
        // According to the spec this could only happen if the file
        // protocol were not recognized.
        throw new Error("unexpected MalformedURLException");
    }
}
}
}

```

XML Class Generator for Java, DTD Example 1b: DTD Input — widl.dtd

The following example, `widl.dtd`, is the DTD file used by `SampleMain.java`.

```

<!ELEMENT WIDL ( SERVICE | BINDING )* >
<!ATTLIST WIDL

```



```
    NAME          CDATA #IMPLIED
    VERSION (1.0 | 2.0 | ...) "2.0"
    BASEURL      CDATA #IMPLIED
    OBJMODEL (wmdom | ...) "wmdom"
>

<!ELEMENT SERVICE EMPTY>
<!ATTLIST SERVICE
    NAME          CDATA #REQUIRED
    URL           CDATA #REQUIRED
    METHOD (Get | Post) "Get"
    INPUT        CDATA #IMPLIED
    OUTPUT       CDATA #IMPLIED
>

<!ELEMENT BINDING ( VARIABLE | CONDITION | REGION ) * >
<!ATTLIST BINDING
    NAME          CDATA #REQUIRED
    TYPE (Input | Output) "Output"
>

<!ELEMENT VARIABLE EMPTY>
<!ATTLIST VARIABLE
    NAME          CDATA #REQUIRED
    TYPE (String | String1 | String2) "String"
    USAGE (Function | Header | Internal) "Function"
    VALUE        CDATA #IMPLIED
    MASK         CDATA #IMPLIED
    NULLOK      (True | False) #REQUIRED
>

<!ELEMENT CONDITION EMPTY>
<!ATTLIST CONDITION
    TYPE (Success | Failure | Retry) "Success"
    REF          CDATA #REQUIRED
    MATCH       CDATA #REQUIRED
    SERVICE     CDATA #IMPLIED
>

<!ELEMENT REGION EMPTY>
<!ATTLIST REGION
    NAME          CDATA #REQUIRED
    START        CDATA #REQUIRED
    END          CDATA #REQUIRED
>
```

XML Class Generator for Java, DTD Example 1c: Input — widl.xml

This XML file inputs SampleMain.java and is based on widl.dtd:

```
<?xml version="1.0"?>
<!DOCTYPE WIDL SYSTEM "Widl.dtd">
<WIDL>
  <SERVICE NAME="sname" URL="surl"/>
  <BINDING NAME="bname"/>
</WIDL>
```

XML Class Generator for Java, DTD Example 1d: TestWidl.java

TestWidl.java constructs an XML document using the Java source files generated by SampleMain.java.

```
/**
 * This is a sample application program which is built using the
 * classes generated by the XML DTD Class Generator. The External DTD
 * File "Widl.dtd" or the XML document which "Widl.xml" which is compliant
 * to Widl.dtd is used to generate the classes. The application
 * SampleMain.java is used to generate the classes which takes the DTD
 * or XML document as input parameters to generate classes.
 */

import oracle.xml.classgen.CGNode;
import oracle.xml.classgen.CGDocument;
import oracle.xml.classgen.DTDClassGenerator;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.DTD;

public class TestWidl
{
    public static void main (String args[])
    {
        try
        {
            WIDL w1 = new WIDL();
            DTD dtd = w1.getDTDNode();

            w1.setName("WIDL1");
            w1.setVersion(WIDL.VERSION_1_0);

            SERVICE s1 = new SERVICE("Service1", "Service_URL");
            s1.setInput("File");
            s1.setOutput("File");
        }
    }
}
```

```
BINDING b1 = new BINDING("Binding1");
b1.setType(BINDING.TYPE_INPUT);

BINDING b2 = new BINDING("Binding2");
b2.setType(BINDING.TYPE_OUTPUT);

VARIABLE v1 = new VARIABLE("Variable1", VARIABLE.NULLOK_FALSE);
v1.setType(VARIABLE.TYPE_STRING);
v1.setUsage(VARIABLE.USAGE_INTERNAL);
v1.setValue("value");

VARIABLE v2 = new VARIABLE("Variable2", VARIABLE.NULLOK_TRUE);
v2.setType(VARIABLE.TYPE_STRING1);
v2.setUsage(VARIABLE.USAGE_HEADER);

VARIABLE v3 = new VARIABLE("Variable3", VARIABLE.NULLOK_FALSE);
v3.setType(VARIABLE.TYPE_STRING2);
v3.setUsage(VARIABLE.USAGE_FUNCTION);
v3.setMask("mask");

CONDITION c1 = new CONDITION("CRef1", "CMatch1");
c1.setService("Service1");
c1.setType(CONDITION.TYPE_SUCCESS);

CONDITION c2 = new CONDITION("CRef2", "CMatch2");
c2.setType(CONDITION.TYPE_RETRY);

CONDITION c3 = new CONDITION("CRef3", "CMatch3");
c3.setService("Service3");
c3.setType(CONDITION.TYPE_FAILURE);

REGION r1 = new REGION("Region1", "Start", "End");

b1.addNode(r1);
b1.addNode(v1);
b1.addNode(c1);
b1.addNode(v2);

b2.addNode(c2);
b2.addNode(v3);

w1.addNode(s1);
w1.addNode(b1);
w1.addNode(b2);
```

```

        wl.validateContent();
        wl.print(System.out);
    }
    catch (Exception e)
    {
        System.out.println(e.toString());
        e.printStackTrace();
    }
}
}

```

XML Class Generator for Java, DTD Example 1e: XML Output — widl.out

This XML file, widl.out, is constructed and printed by TestWidl.java.

```

<?xml version = '1.0' encoding = 'ASCII'?>
<!DOCTYPE WIDL SYSTEM "file:/oracore/java/xml/ORACORE_MAIN_SOLARIS_990115_
XMLCLASSGEN/sample/out/WIDL.dtd">
<WIDL NAME="WIDL1" VERSION="1.0">
    <SERVICE NAME="Service1" URL="Service_URL" INPUT="File" OUTPUT="File"/>
    <BINDING NAME="Binding1" TYPE="Input">
        <REGION NAME="Region1" START="Start" END="End" />
        <VARIABLE NAME="Variable1" NULLOK="False" TYPE="String" USAGE="Internal"
VALUE="value"/>
        <CONDITION REF="CRef1" MATCH="CMatch1" SERVICE="Service1" TYPE="Success"/>
        <VARIABLE NAME="Variable2" NULLOK="True" TYPE="String1" USAGE="Header"/>
    </BINDING>
    <BINDING NAME="Binding2" TYPE="Output">
        <CONDITION REF="CRef2" MATCH="CMatch2" TYPE="Retry"/>
        <VARIABLE NAME="Variable3" NULLOK="False" TYPE="String2" USAGE="Function"
MASK="mask"/>
    </BINDING>
</WIDL>

```

The following Class Generator using XML Schema examples are included here

- [XML Class Generator for Java, Schema Example 1a: XML Schema, car.xsd](#)
- [XML Class Generator for Java, Schema Example 1b: Application, CarDealer.java](#)
- [XML Class Generator for Java, Schema Example 2a: Schema — book.xsd](#)
- [XML Class Generator for Java, Schema Example 2b: BookCatalogue.java](#)
- [XML Class Generator for Java, Schema Example 3a: Schema — po.xsd](#)
- [XML Class Generator for Java, Schema Example 3b: Application — TestPo.java](#)

XML Class Generator for Java, Schema Example 1a: XML Schema, car.xsd

This sample, car.xsd, is used in an `oracg` command to generate classes. These classes inputs the program, `CarDealer.java`, which then creates an XML document. The command used is:

```
oracg -c -s car.xsd -p package1
```

See the comments about how this is used, in:

- ["XML Class Generator for Java, Schema Example 1b: Application, CarDealer.java" on page 22-20](#)
- ["Running XML Class Generator for Java — XML Schema Examples" on page 22-11](#)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
targetNamespace = "http://www.CarDealers.com/" elementFormDefault="qualified">
<element name="Car">
  <complexType>
    <element name="Model">
      <simpleType base="string">
        <enumeration value = "Ford"/>
        <enumeration value = "Saab"/>
        <enumeration value = "Audi"/>
      </simpleType>
    </element>
    <element name="Make">
      <simpleType base="string">
        <minLength value = "1"/>
        <maxLength value = "30"/>
      </simpleType>
    </element>
    <element name="Year">
      <complexType content="mixed">
        <attribute name="PreviouslyOwned" type="string" use="required"/>
        <attribute name="YearsOwned" type="integer" use="optional"/>
      </complexType>
    </element>
    <element name="OwnerName" type="string" minOccurs="0"
maxOccurs="unbounded"/>
    <element name="Condition">
      <complexType base="string" derivedBy="extension">
        <attribute name="Automatic">
          <simpleType base="string">
            <enumeration value = "Yes"/>
          </simpleType>
        </attribute>
      </complexType>
    </element>
  </complexType>
</element>
```

```
        <enumeration value = "No"/>
    </simpleType>
</attribute>
</complexType>
</element>
<element name="Mileage">
    <simpleType base="integer">
        <minInclusive value="0"/>
        <maxInclusive value="20000"/>
    </simpleType>
</element>
<attribute name="RequestDate" type="date"/>
</complexType>
</element>
</schema>
```

XML Class Generator for Java, Schema Example 1b: Application, CarDealer.java

```
/**
 * This is a sample application program that creates an XML document. It is
 * built using the classes generated by XML Schema Class Generator. XML
 * Schema "car.xsd", is used to generate the classes using the oracy
 * command line utility. The classes are generated in a package called
 * packagel which is specified as command line option. The following
 * oracy command line options are used to generate the classes:
 * oracy -c -s car.xsd -p packagel
 */

import oracle.xml.classgen.CGXSElement;
import oracle.xml.classgen.SchemaClassGenerator;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.XMLOutputStream;
import java.io.OutputStream;

import packagel.*;

public class CarDealer
{
    static OutputStream output = System.out;
    static XMLOutputStream out = new XMLOutputStream(output);

    public static void main(String args[])
    {
        CarDealer cardealer = new CarDealer();
        try
```

```
{
    Car.Car_Type ctype = new Car.Car_Type();
    ctype.setRequestDate("02-09-00");
    Car.Car_Type.Model model = new Car.Car_Type.Model();
    Car.Car_Type.Model.Model_Type modelType =
        new Car.Car_Type.Model.Model_Type("Ford");
    model.setType(modelType);
    ctype.addModel(model);

    Car.Car_Type.Make make = new Car.Car_Type.Make();
    Car.Car_Type.Make.Make_Type makeType =
        new Car.Car_Type.Make.Make_Type("F150");
    make.setType(makeType);
    ctype.addMake(make);

    Car.Car_Type.Year year = new Car.Car_Type.Year();
    Car.Car_Type.Year.Year_Type yearType =
        new Car.Car_Type.Year.Year_Type();
    yearType.addText("1999");

    year.setType(yearType);
    ctype.addYear(year);

    Car.Car_Type.OwnerName owner1 = new Car.Car_Type.OwnerName();
    owner1.setType("Joe Smith");
    ctype.addOwnerName(owner1);

    Car.Car_Type.OwnerName owner2 = new Car.Car_Type.OwnerName();
    owner2.setType("Bob Smith");
    ctype.addOwnerName(owner2);

    String str = "Small dent on the car's right bumper.";
    Car.Car_Type.Condition condition = new Car.Car_Type.Condition();
    Car.Car_Type.Condition.Condition_Type conditionType =
        new Car.Car_Type.Condition.Condition_Type(str);

    Car.Car_Type.Condition.Condition_Type.Automatic automatic =
        new Car.Car_Type.Condition.Condition_Type.Automatic("Yes");
    conditionType.setAutomatic(automatic);

    condition.setType(conditionType);
    ctype.addCondition(condition);

    Car.Car_Type.Mileage mileage = new Car.Car_Type.Mileage();
    Car.Car_Type.Mileage.Mileage_Type mileageType =
```

```
        new Car.Car_Type.Mileage.Mileage_Type("10000");
mileage.setType(mileageType);
ctype.addMileage(mileage);

Car car = new Car();
car.setType(ctype);
car.print(out);

out.writeNewLine();
out.flush();
    }
catch(InvalidContentException e)
{
    System.out.println(e.getMessage());
    e.printStackTrace();
}
catch(Exception e)
{
    System.out.println(e.getMessage());
    e.printStackTrace();
}
    }
}
```

XML Class Generator for Java, Schema Example 2a: Schema — book.xsd

This sample schema, `book.xsd`, is used in an `oracg` command to generate classes. The classes then input the program, `CarDealer.java`, which creates an XML document. The `oracg` command is:

```
oracg -c -s book.xsd -p package2
```

See the comments about how this is used, in:

- ["XML Class Generator for Java, Schema Example 2b: BookCatalogue.java"](#) on page 22-23
- ["Running XML Class Generator for Java — XML Schema Examples"](#) on page 22-11

```
<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.somewhere.org/BookCatalogue"
        xmlns:cat = "http://www.somewhere.org/BookCatalogue"
        elementFormDefault="qualified">
```



```

<complexType name="Pub">
  <sequence>
    <element name="Title" type="cat:titleType" maxOccurs="*" />
    <element name="Author" type="string" maxOccurs="*" />
    <element name="Date" type="date" />
  </sequence>
  <attribute name="language" type="string" use="default" value="English" />
</complexType>

<complexType name="titleType" base="string" derivedBy="extension">
  <attribute name="old" type="string" use="default" value="false" />
</complexType>

  <element name="Catalogue" type="cat:Pub" />
</schema>

```

XML Class Generator for Java, Schema Example 2b: BookCatalogue.java

```

/**
 * This is a sample application program built using the
 * classes generated by XML Schema Class Generator. XML
 * Schema "book.xsd" is used to generate the classes using the oracy
 * command line utility. The classes are generated in a package called
 * package2 which is specified as command line option. The following
 * oracy command line options are used to generate the classes:
 * oracy -c -s book.xsd -p package2
 */

import oracle.xml.classgen.SchemaClassGenerator;
import oracle.xml.classgen.CGXSElement;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.XMLOutputStream;
import java.io.OutputStream;

import package2.*;

public class BookCatalogue
{
  static OutputStream output = System.out;
  static XMLOutputStream out = new XMLOutputStream(output);

  public static void main(String args[])
  {

```

```
BookCatalogue bookCatalogue = new BookCatalogue();
try
{
    Pub pubType = new Pub();

    TitleType titleType = new TitleType("Natural Health");
    titleType.setOld("true");

    Pub.Title title = new Pub.Title();
    title.setType(titleType);
    pubType.addTitle(title);

    Pub.Author author = new Pub.Author();
    author.setType("Richard> Bach");
    pubType.addAuthor(author);

    Pub.Date date = new Pub.Date();
    date.setType("1977");
    pubType.addDate(date);
    pubType.setLanguage("English");

    Catalogue catalogue = new Catalogue();
    catalogue.setType(pubType);

    catalogue.print(out);
    out.writeNewLine();
    out.flush();
}
catch(InvalidContentException e)
{
    System.out.println(e.getMessage());
    e.printStackTrace();
}
catch(Exception e)
{
    System.out.println(e.getMessage());
    e.printStackTrace();
}
}
```

XML Class Generator for Java, Schema Example 3a: Schema — po.xsd

This sample schema, `po.xsd`, is used in an `oracg` command to generate classes. The classes then input the program, `TestPo.java`, which creates an XML document. The `oracg` command used is:

```
oracg -c -s po.xsd -p package3
```

See the comments about how this is used, in:

- ["XML Class Generator for Java, Schema Example 3b: Application — TestPo.java" on page 22-26](#)
- ["Running XML Class Generator for Java — XML Schema Examples" on page 22-11](#)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.somewhere.org/PurchaseOrder"
        xmlns:po = "http://www.somewhere.org/PurchaseOrder">

<element name="comment" type="string"/>

<element name="PurchaseOrder">
  <complexType>
    <element name="shipTo" type="po:Address"/>
    <element name="billTo" type="po:Address"/>
    <element ref="po:comment" minOccurs="0"/>
    <element name="items" type="po:Items"/>
    <attribute name="orderDate" type="date"/>
    <attribute name="shipDate" type="date"/>
    <attribute name="receiveDate" type="date"/>
  </complexType>
</element>

<complexType name="Address">
  <element name="name" type="string"/>
  <element name="street" type="string"/>
  <element name="city" type="string"/>
  <element name="zip" type="decimal"/>
  <attribute name="country" type="NMTOKEN"
    use="fixed" value="US"/>
</complexType>

<complexType name="Items">
```

```

<element name="item" minOccurs="0" maxOccurs="unbounded">
  <complexType>
    <element name="productName" type="string"/>
    <element name="quantity" type="int"/>
    <element name="price" type="decimal"/>
    <element name="shipDate" type="date" minOccurs='0'/>
    <attribute name="partNum" type="string"/>
  </complexType>
</element>
</complexType>

</schema>

```

XML Class Generator for Java, Schema Example 3b: Application — TestPo.java

```

/**
 * This is a sample application program which is built using the
 * classes generated by XML Schema Class Generator. XML
 * Schema "po.xsd" is used to generate the classes using the oracy
 * command line utility. The classes are generated in a package called
 * package3 which is specified as command line option. The following
 * oracy command line options are used to generate the classes:
 * oracy -c -s po.xsd -p package3
 */

import oracle.xml.classgen.CGXSElement;
import oracle.xml.classgen.SchemaClassGenerator;
import oracle.xml.classgen.InvalidContentException;
import oracle.xml.parser.v2.XMLOutputStream;
import java.io.OutputStream;
import package3.*;

public class TestPo
{
    static OutputStream output = System.out;
    static XMLOutputStream out = new XMLOutputStream(output);

    public static void main (String args[])
    {
        TestPo testpo = new TestPo();
        try
        {
            // Create Purchase Order
            PurchaseOrder po = new PurchaseOrder();

```

```
// Create Purchase Order Type
PurchaseOrder.PurchaseOrder_Type poType =
    new PurchaseOrder.PurchaseOrder_Type();

// Set purchase order date
poType.setOrderDate("December 17, 2000");
poType.setShipDate("December 19, 2000");
poType.setReceiveDate("December 21, 2000");

// Create a PurchaseOrder shipTo item
PurchaseOrder.PurchaseOrder_Type.ShipTo shipTo =
    new PurchaseOrder.PurchaseOrder_Type.ShipTo();

// Create Address
Address address = new Address();

// Create the Name for the address and add
// it to addresss
Address.Name name = new Address.Name();
name.setType("Mary Smith");
address.addName(name);

// Create the Stree name for the address and add
// it to the address
Address.Street street = new Address.Street();
street.setType("Laurie Meadows");
address.addStreet(street);

// Create the city name for the address and add
// it to the address
Address.City city = new Address.City();
city.setType("San Mateo");
address.addCity(city);

// Create the zip name for the address and add
// it to the address
Address.Zip zip = new Address.Zip();
zip.setType(new Double("11208"));
address.addZip(zip);

// Set the address of the shipTo object
shipTo.setType(address);
// Add the shipTo to the Purchase Type object
poType.addShipTo(shipTo);
```

```
// Create a Purchase Order BillTo item
PurchaseOrder.PurchaseOrder_Type.BillTo billTo =
    new PurchaseOrder.PurchaseOrder_Type.BillTo();

// Create a billing Address
Address billingAddress = new Address();

// Create the name for billing address, set the
// name and add it to the billing address
Address.Name name1 = new Address.Name();
name1.setType("John Smith");
billingAddress.addName(name1);

// Create the street name for the billing address,
// set the street name value and add it to the
// billing address
Address.Street street1 = new Address.Street();
street1.setType("No 1. North Broadway");
billingAddress.addStreet(street1);

// Create the City name for the address, set the
// city name value and add it to the billing address
Address.City city1 = new Address.City();
city1.setType("New York");
billingAddress.addCity(city1);

// Create the Zip for the address, set the zip
// value and add it to the billing address.
Address.Zip zip1 = new Address.Zip();
zip1.setType(new Double("10006"));
billingAddress.addZip(zip1);

// Set the type of the billTo object to billingAddress
billTo.setType(billingAddress);

// Add the billing address to the PurchaseOrder type
poType.addBillTo(billTo);

PurchaseOrder.PurchaseOrder_Type.Items pItem =
    new PurchaseOrder.PurchaseOrder_Type.Items();

Items items = new Items();
Items.Item item = new Items.Item();
Items.Item.Item_Type itemType = new Items.Item.Item_Type();
```

```
Items.Item.Item_Type.ProductName pname =
    new Items.Item.Item_Type.ProductName();
pname.setType("Perfume");
itemType.addProductName(pname);

Items.Item.Item_Type.Quantity qty =
    new Items.Item.Item_Type.Quantity();
qty.setType(new Integer("1"));
itemType.addQuantity(qty);

Items.Item.Item_Type.Price price =
    new Items.Item.Item_Type.Price();
price.setType(new Double("69.99"));
itemType.addPrice(price);

Items.Item.Item_Type.ShipDate sdate =
    new Items.Item.Item_Type.ShipDate();
sdate.setType("Feb 14. 2000");
itemType.addShipDate(sdate);

itemType.setPartNum("ITMZ411");

item.setType(itemType);
items.addItem(item);

pItem.setType(items);

poType.addItems(pItem);

// Set the type of the Purchase Order object to
// Purchase Order Type
po.setType(poType);
po.print(out);

out.writeNewLine();
out.flush();
}
catch (InvalidContentException e)
{
    System.out.println(e.getMessage());
    e.printStackTrace();
}
catch (Exception e)
{
    System.out.println(e.toString());
```

```
        e.printStackTrace();
    }
}
```

Frequently Asked Questions (FAQs): Class Generator for Java

This section lists XML Java Class Generator questions and answers.

How Do I Install XML Class Generator?

Question

How do i install XML Java Class Generator?

Answer

The Class Generator is packaged as part of the XDK and so you do not have to download it separately. The CLASSPATH should be set to include classgen.jar, xmlparserv2.jar, and xschema.jar which are located in the lib/ directory and not in the bin/ directory.

What Does XML Class Generator for Java Do?

Question

What does the XML Class Generator for Java do? How do I use the XML Class Generator for Java to get XML data?

Answer

XML Class Generator for Java creates Java source files from an XML DTD. This is useful when an application wants to send an XML message to another application based on an agreed-upon DTD or as the back end of a web form to construct and XML document. Using these classes, Java applications can construct, validate, and print XML documents that comply with the input DTD. The Class Generator works in conjunction with the Oracle XML Parser for Java v2, which parses the DTD and passes the parsed document to the class generator.

To get XML data, first, get the data from the database using JDBC ResultSets. Then, instantiate objects using the classes generated by the XML Class Generator.

Which DTD's are Supported?

Question

Does XML Java Class Generator support any kind of DTD?

Answer

Yes, it supports any kind of DTD that is XML 1.0 compliant.

The Classes Not Found Error When Running XML Class Generator Samples?

Question

How do I fix the classes-not-found errors?

Answer

Correct your CLASSPATH to include classgen.jar, xmlparserv2.jar, and xschema.jar.

In XML Class Generator, How Do I Create the Root Object More than Once?

Question

I've generated, from a DTD, a set of Java classes with "ClassGenerator". After, I've tried to create a Java application that uses these classes to create an XML file from data passed as arguments. I cannot create the root object, the object derived from CGDocument, more than one time because I obtain the following error message:

```
oracle.xml.parser.XMLDOMException: Node doesn't belong to the current document
```

How do I handle the star operator (*). When the application starts I do not know how many times the element will appear. Thus I do not build a static loop where I make a sequence of "element.addNode()". The problem is that some of these will be empty and I will obtain an XML document with a set of empty elements with empty attributes.

Answer

You can create subsequent XML docs by calling the constructor each time. A well-formed XML document is not permitted to have more than one root node, therefore you cannot use the "*" on the element you are designating as the document root.

How Can I Create XML Files from Scratch Using the DOM API?

Question

I want to create an XML file using the DOM API. I do not want to create the XML file by typing in the text editor

```
<xml>
  <future>is great</future>
</xml>
```

instead, I want to create it using the DOM API. There are several examples of manipulating an XML file using the DOM once there is an input file, but not the other way round. That is, of creating the XML file from scratch (when there is no input file) using the DOM, once you know the “tagnames” and their “values”.

Answer

The simplest way is download XML Class Generator for Java and give it a DTD of the XML document you want. It will create the DOM classes to programmatically create XML documents. There are samples included with the software.

Can I Create an XML Document in a Java Class?

Question

I need to create an XML document in a java class as follows

```
<?xml version = '1.0' encoding = 'WINDOWS-1252'?>
  <root>
    <listing>
      <one> test </one>
      <two> test </two>
    </listing>
  </root>
```

Can I use the XMLDocument class to create an XML document? I know about the XML SQL Utility, but that only creates XML based on SQL queries which is not what I am after on this occasion. Do you have an example of how to do this?

Answer

XML Class Generator, available from <http://otn.oracle.com/tech/xml> as part of the Oracle XDK for Java, does the job nicely. The XDKs are also available with Oracle

and Oracle Application Server products. The Class Generator generates Java classes for each element in your DTD. These classes can then be used to dynamically construct an XML document at runtime. The Class Generator download contains sample code.

Part VII

XDK for Java Beans

Part VII describes how to use XML Developer's Kit (XDK) for Java Beans.

Part VII contains the following chapter:

- [Chapter 23, "Using XML Transviewer Beans"](#)

Using XML Transviewer Beans

This chapter contains the following sections:

- [Accessing Oracle XML Transviewer Beans](#)
- [XDK for Java: XML Transviewer Bean Features](#)
- [Using the XML Transviewer Beans](#)
- [Using XMLSourceView Bean](#)
- [Using XMLTransformPanel Bean](#)
- [Using XSLTransformer Bean](#)
- [Using DOMBuilder Bean](#)
- [Using Treeviewer Bean](#)
- [Using DBViewer Bean](#)
- [Using DBAccess Bean](#)
- [Installing the Transviewer Bean Samples](#)
- [Transviewer Bean Example 1: AsyncTransformSample.java](#)
- [Transviewer Bean Example 2: ViewSample.java](#)
- [Transviewer Bean Example 3: XMLTransformPanelSample.java](#)
- [Transviewer Bean Example 4a: DBViewer Bean — DBViewClaims.java](#)
- [Transviewer Bean Example 4b: DBViewer Bean — DBViewFrame.java](#)
- [Transviewer Bean Example 4c: DBViewer Bean — DBViewSample.java](#)

Accessing Oracle XML Transviewer Beans

The Oracle XML Transviewer beans are provided with Oracle Enterprise and Standard Editions from Release 2 (8.1.6) and higher, as part of XDK for Java Beans. If you do not have these editions you can download the beans from the site: <http://otn.oracle.com/tech/xml>

XDK for Java: XML Transviewer Bean Features

XML Transviewer Beans facilitate the addition of graphical or visual interfaces to your XML applications.

Direct Access from JDeveloper

Bean encapsulation includes documentation and descriptors that can be accessed directly from Java Integrated Development Environments like JDeveloper.

Sample Transviewer Bean Application is Included

A sample application that demonstrates all of the beans to create a simple XML editor and XSL transformer is included with your software.

The included sample application with XML SQL Utility (XSU) cause the following:

- Database queries to materialize XML
- Transform the XML through an XSL stylesheet
- Store the resulting XML document back in the database for fast retrieval

Database Connectivity

Database Connectivity is included with the XML Transviewer Beans. The beans can now connect directly to a JDBC-enabled database to retrieve and store XML and XSL files.

XML Transviewer Beans

XML Transviewer Beans are comprised of the following beans:

DOMBuilder Bean

The DOMBuilder bean is a non-visual bean. It builds a DOMTree from an XML document.

DOM Builder bean encapsulates the XML Parser for Java's DOMParser class with a bean interface, and extends its functionality to permit asynchronous parsing. By registering a listener, Java applications can parse large or successive documents having control return immediately to the caller. See "[Using DOMBuilder Bean](#)" on page 23-5.

XSLTransformer Bean

The XSLTransformer bean is a non-visual bean. It accepts an XML file, applies the transformation specified by an input XSL stylesheet, and creates the resulting output file.

XSLTransformer bean enables you to transform an XML document to almost any text-based format including XML, HTML and DDL, by applying the appropriate XSL stylesheet.

- When integrated with other beans, XSLTransformer bean enables an application or user to view the results of transformations immediately.
- This bean can also be used as the basis of a server-side application or servlet to render an XML document, such as an XML representation of a query result, into HTML for display in a browser.

See "[Using XSLTransformer Bean](#)" on page 23-9.

Treeviewer Bean

The Treeviewer bean displays XML formatted files graphically as a tree. The branches and leaves of this tree can be manipulated with a mouse. See "[Using Treeviewer Bean](#)" on page 23-13.

XMLSourceView Bean

The XMLSourceView bean is a visual Java bean. It allows visualization of XML documents and editing. It enables the display of XML and XSL formatted files with color syntax highlighting when modifying an XML document with an editing application. This helps view and edit the files. It can be easily integrated with DOM Builder bean, and allows for pre or post parsing visualization, and validation against a specified DTD. See "[Using XMLSourceView Bean](#)" on page 23-15.

XMLTransformPanel Bean

A visual Java bean that applies XSL transformations on XML documents and shows the results. It allows editing of XML and XSL input files. See "[Using XMLTransformPanel Bean](#)" on page 23-20.

DBViewer Bean

DBViewer bean is 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. DBViewer bean has XML and XSL tree buffers as well as a result buffer. DBViewer bean allows the calling program to do the following:

- Load or save buffers from various sources such as from CLOB tables in an Oracle database or from the file system. Control can be also used to move files between the file system and the user schema in the database.
- 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 a source or tree structure. The *result* buffer content can be rendered as HTML and also shown as source or tree structure. The XML buffer can be loaded from a database query.

DBAccess Bean

DBAccess bean maintains CLOB tables that hold multiple XML and text documents.

Using the XML Transviewer Beans

Guidelines for using the XML Transviewer Beans are described in the following sections:

- [Using DOMBuilder Bean](#)
- [Using XSLTransformer Bean](#)
- [Using Treeviewer Bean](#)
- [Using XMLSourceView Bean](#)
- [Using XMLTransformPanel Bean](#)
- [Using DBViewer Bean](#)
- [Using DBAccess Bean](#)

See Also:

- Oracle9i XML Reference
- [Chapter 3, "Oracle XML Developer Kits \(XDKs\) and Components: Overview and General FAQs"](#).

Using DOMBuilder Bean

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

The parsing is done in a separate thread and DOMBuilderInterface interface must be used for notification when the tree is built.

Used for Asynchronous Parsing in the Background

DOM Builder bean encapsulates XML Parser for Java with a bean interface. It extends its functionality to permit asynchronous parsing. By registering a listener, a Java application can parse documents and return control return to the caller.

Asynchronous parsing in a background thread can be used interactively in visual applications. For example, when parsing a large file with the normal parser, the user interface can freeze till the parsing has completed. This can be avoided with the DOMBuilder bean. After calling the DOMBuilder bean parse method, the application can receive control back immediately and display “Parsing, please wait”. If a “Cancel” button is included you can also cancel the operation. The application can continue when `domBuilderOver()` method is called by DOMBuilder bean when background parsing task has completed.

DOMBuilder Bean Parses Many Files Fast

When parsing a large number of files, DOMBuilder bean can save you much time. Up to 40% faster times have been recorded when compared to parsing the files one by one.

DOMBuilder Bean Usage

Figure 23–1 illustrates DOMBuilder Bean usage.

1. The XML document to be parsed is input as a file, string buffer, or URL.
2. This inputs
`DOMBuilder.addDOMBuilderInterface(DOMBuilderInterface)` method.
This adds DOMBuilderInterface.
3. The `DOMBuilder.parser()` method parses the XML document.
4. Optionally, the parsed result undergoes further processing. See Table 23–1 for a list of available methods to apply.

5. `DOMBuilderListener` API is called using `DOMBuilderOver()` method. This is called when it received an async call from an application. This interface must be implemented to receive notifications about events during asynchronous parsing. The class implementing this interface must be added to the `DOMBuilder` using `addDOMBuilderListener` method.

Available `DOMBuilderListener` methods are:

- `domBuilderError(DOMBuilderEvent)`. This method is called when parse error occur.
 - `domBuilderOver(DOMBuilderEvent)`. This method is called when the parse is complete
 - `domBuilderStarted(DOMBuilderEvent)`. This method is called when parsing starts
6. `DOMBuilder.getDocument()` fetches the resulting DOM document and outputs the DOM document.

Figure 23-1 DOMBuilder Bean Usage

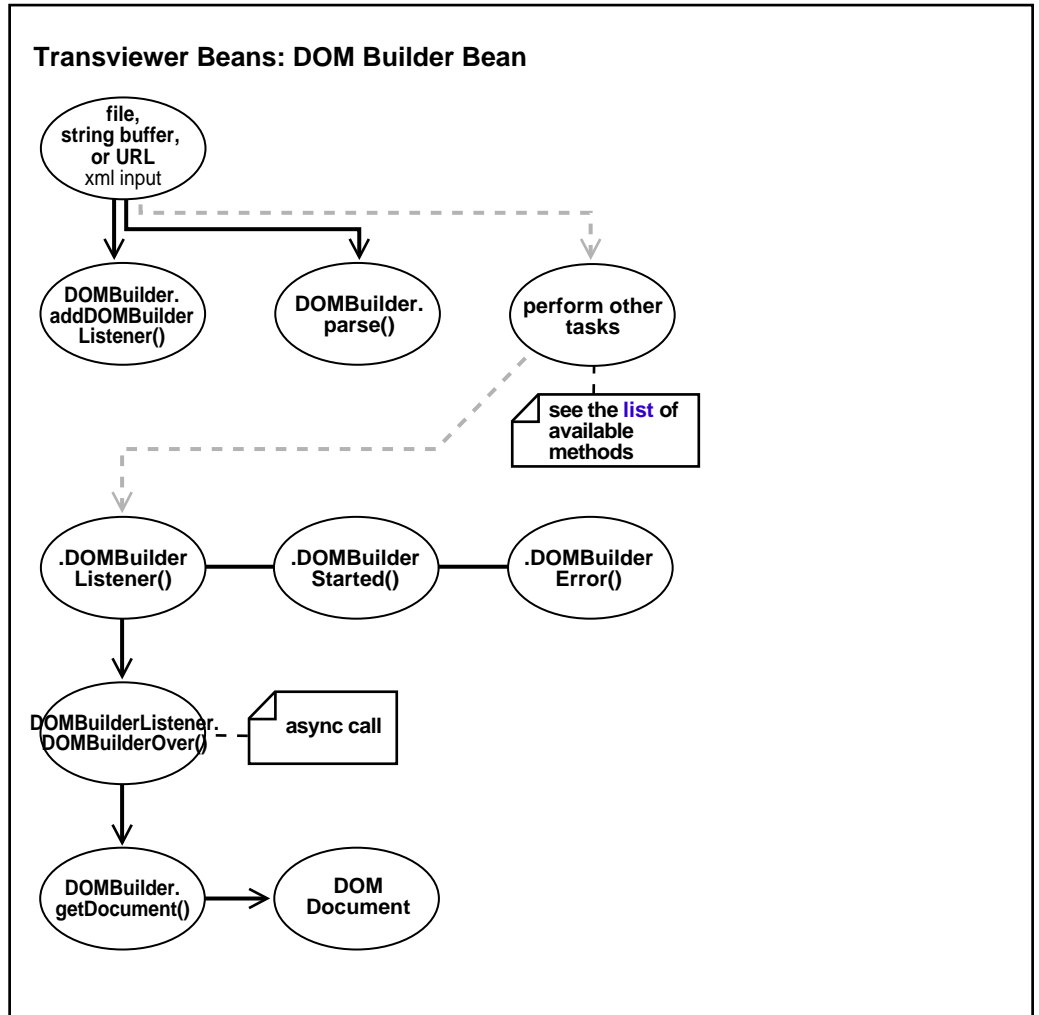


Table 23–1 DOMBuilder Bean: Methods

Method	Description
addDOMBuilderErrorListener(DOMBuilderErrorListener)	Adds DOMBuilderErrorListener
addDOMBuilderListener(DOMBuilderListener)	Adds DOMBuilderListener
	Get the DTD
getDocument()	Gets the document
getId()	Returns the parser object id.
getReleaseVersion()	Returns the release version of the Oracle XML Parser
	Gets the document
getValidationMode()	Returns the validation mode
parse(InputSource)	Parses the XML from given input source
	Parses the XML from given input stream.
parse(Reader)	Parses the XML from given input stream.
parse(String)	Parses the XML from the URL indicated
parse(URL)	Parses the XML document pointed to by the given URL and creates the corresponding XML document hierarchy.
parseDTD(InputSource, String)	Parses the XML External DTD from given input source
parseDTD(InputStream, String)	Parses the XML External DTD from given input stream.
parseDTD(Reader, String)	Parses the XML External DTD from given input stream.
	Parses the XML External DTD from the URL indicated
parseDTD(URL, String)	Parses the XML External DTD document pointed to by the given URL and creates the corresponding XML document hierarchy.
removeDOMBuilderErrorListener(DOMBuilderErrorListener)	Removes DOMBuilderErrorListener
removeDOMBuilderListener(DOMBuilderListener)	Removes DOMBuilderListener
run()	This method runs in a thread

Table 23–1 DOMBuilder Bean: Methods (Cont.)

Method	Description
	Set the base URL for loading external entities and DTDs.
setDebugMode(boolean)	Sets a flag to turn on debug information in the document
setDoctype(DTD)	Sets the DTD
setErrorStream(OutputStream)	Creates an output stream for the output of errors and warnings.
setErrorStream(OutputStream, String)	Creates an output stream for the output of errors and warnings.
setErrorStream(PrintWriter)	Creates an output stream for the output of errors and warnings.
	Sets the node factory.
setPreserveWhitespace(boolean)	Sets the white space preserving mode
setValidationMode(boolean)	Sets the validation mode
showWarnings(boolean)	Switches to determine whether to print warnings

Using XSLTransformer Bean

The XSLTransformer bean accepts an XML file and applies the transformation specified by an input XSL stylesheet, to create and output file. It enables you to transform an XML document to almost any text-based format including XML, HTML and DDL, by applying an XSL stylesheet.

When integrated with other beans, XSLTransformer bean enables an application or user to view the results of transformations immediately.

This bean can also be used as the basis of a server-side application or servlet to render an XML document, such as an XML representation of a query result, into HTML for display in a browser.

The XSLTransformer bean encapsulates the Java XML Parser XSLT processing engine with a bean interface and extends its functionality to permit asynchronous transformation.

By registering a listener, your Java application can transform large and successive documents by having the control returned immediately to the caller.

Many Files to Transform? Use XSLTransformer Bean

XSL transformations can be time consuming. Use XSL Transformer bean in applications that transform large number of files. It can transform multiple files concurrently.

Need a responsive User Interface? Use XSLTransformer Bean

XSLTransformer bean can be used for visual applications for a responsive user interface. There are similar issues here as with DOMBuilder bean.

By implementing `XSLTransformerListener()` method, the caller application can be notified when the transformation is complete. The application is free to perform other tasks in between requesting and receiving the transformation.

XSL Transviewer Bean Scenario 1: Regenerating HTML — Underlying Data Changes

This scenario illustrates one way of applying XSLTransformer bean.

1. Create a SQL query. Store the selected XML data in a CLOB table.
2. Using the XSLTransformer bean, create an XSL stylesheet and interactively apply this to the XML data until you are satisfied by the data presentation. This can be HTML produced by the XSL transformation.
3. Now that you have the desired SQL (data selection) and XSL (data presentation), create a trigger on the table or view used by your SQL query. The trigger can execute a stored procedure. The stored procedure, can for example, do the following:
 - Run the query
 - Apply the stylesheet
 - Store the resulting HTML in a CLOB table.
4. This process can repeat whenever the source data table is updated.

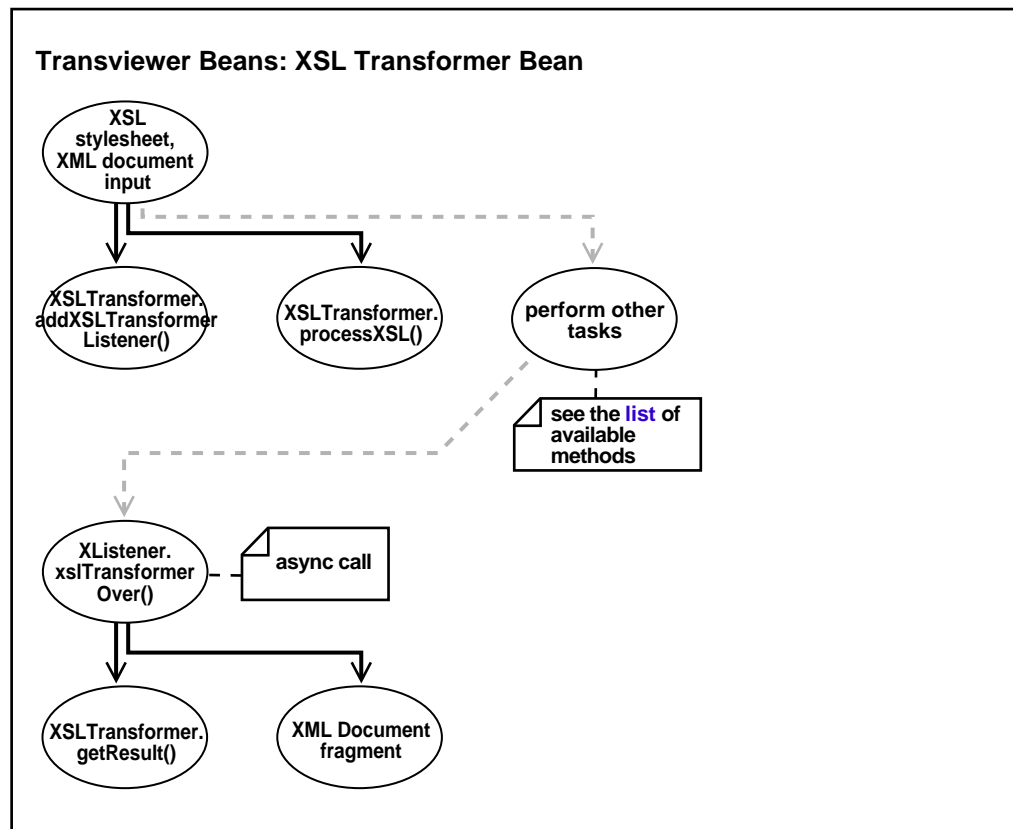
The HTML stored in the CLOB table always mirrors the last data stored in the tables being queried. A JSP (Java Server Page) can display the HTML.

In this scenario, multiple end users do not produce multiple data queries that contribute to bigger loads to the database. The HTML is regenerated only when the underlying data changes and only then.

XSLTransformer Bean Usage

Figure 23–2 illustrates the XSLTransformer bean usage. For examples of implementing this bean, see "Transviewer Bean Example 1: AsyncTransformSample.java".

Figure 23–2 XSLTransformer Bean Usage



1. An XSL stylesheet and XML document input the XSLTransformer using method:
`XSLTransformer.addXSLTransformerListener(XSLTransformerListener)`. This adds a listener.

2. The `XSLTransformer.processXSL()` method initiates the XSL transformation in the background.
3. Optionally, other work can be assigned to the XSLTransformer bean. [Table 23–2](#) lists available XSLTransformer bean methods.
4. When the transformation is complete, an asynchronous call is made and the `XSLTransformerListener.xslTransformerOver()` method is called. 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 event queue using `addXSLTransformerListener` method.
5. `XSLTransformer.getResult()` method returns the XML document fragment for the resulting document.
6. It outputs the XML document fragment.

Table 23–2 XSLTransformer Bean: Methods

Method	Description
<code>addXSLTransformerErrorListener(XSLTransformerErrorListener)</code>	Adds an error event listener
<code>addXSLTransformerListener(XSLTransformerListener)</code>	Adds a listener
<code>getId()</code>	Returns the unique XSLTransformer id
<code>getResult()</code>	Returns the document fragment for the resulting document.
<code>processXSL(XSLStylesheet, InputStream, URL)</code>	Initiates XSL Transformation in the background.
<code>processXSL(XSLStylesheet, Reader, URL)</code>	Initiates XSL Transformation in the background.
<code>processXSL(XSLStylesheet, URL, URL)</code>	Initiates XSL Transformation in the background.
<code>processXSL(XSLStylesheet, XMLDocument)</code>	Initiates XSL Transformation in the background.
<code>processXSL(XSLStylesheet, XMLDocument, OutputStream)</code>	Initiates XSL Transformation in the background.
<code>removeDOMTransformerErrorListener(XSLTransformerErrorListener)</code>	Removes an error event listener
<code>removeXSLTransformerListener(XSLTransformerListener)</code>	Removes a listener
<code>run()</code>	
<code>setErrorStream(OutputStream)</code>	Sets the error stream used by the XSL processor
<code>showWarnings(boolean)</code>	Sets the showWarnings flag used by the XSL processor

Using Treeviewer Bean

The Treeviewer bean displays an XML document as a tree. It recognizes the following XML DOM nodes:

- Tag
- Attribute Name
- Attribute Value
- Comment
- CDATA
- PCDATA
- PI Data
- PI Name
- NOTATION Symbol

It takes as input an `org.w3c.dom.Document` object.

[Figure 23-3, "Treeviewer Bean in Action: Displaying an XML Document as a Tree"](#) shows how the Treeviewer bean displays the XML document and the editing options.

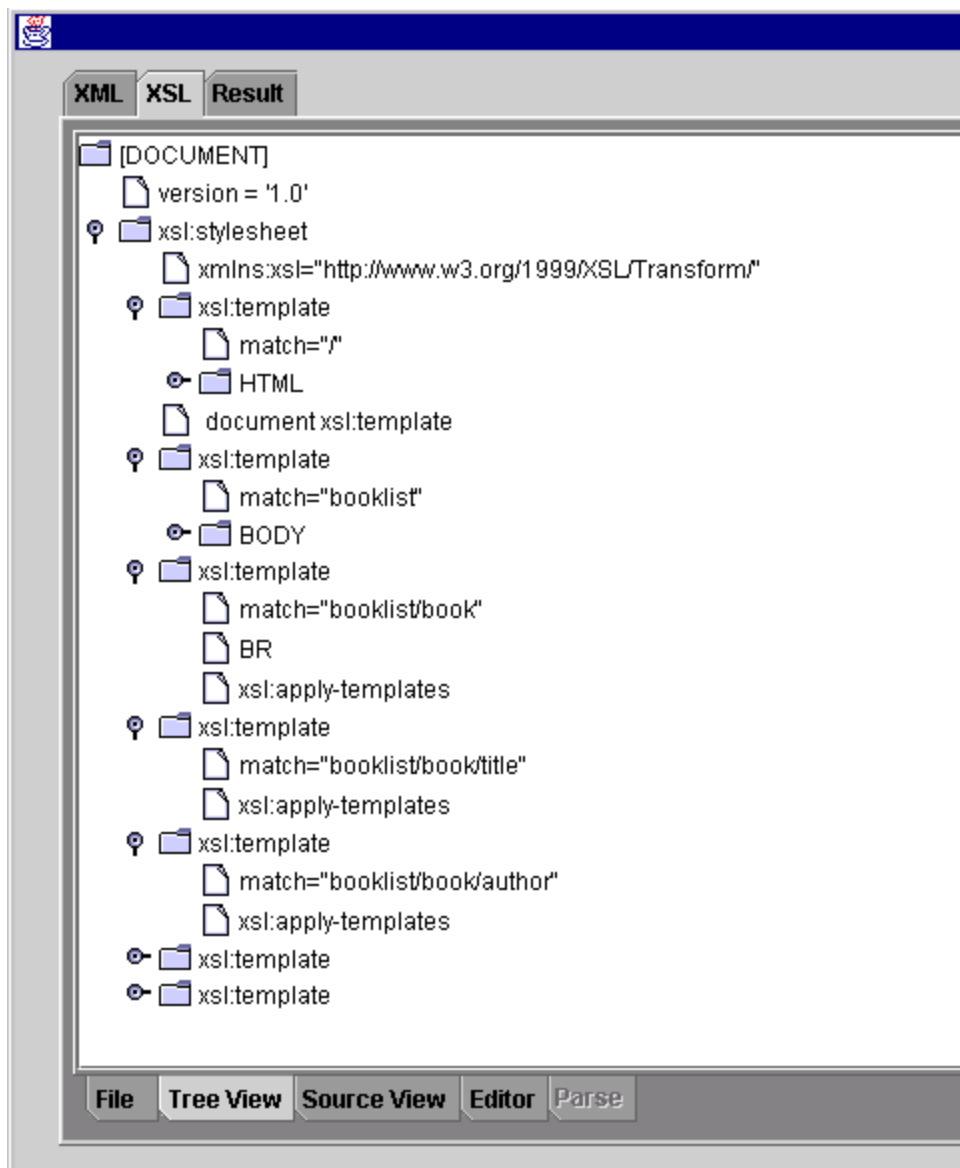
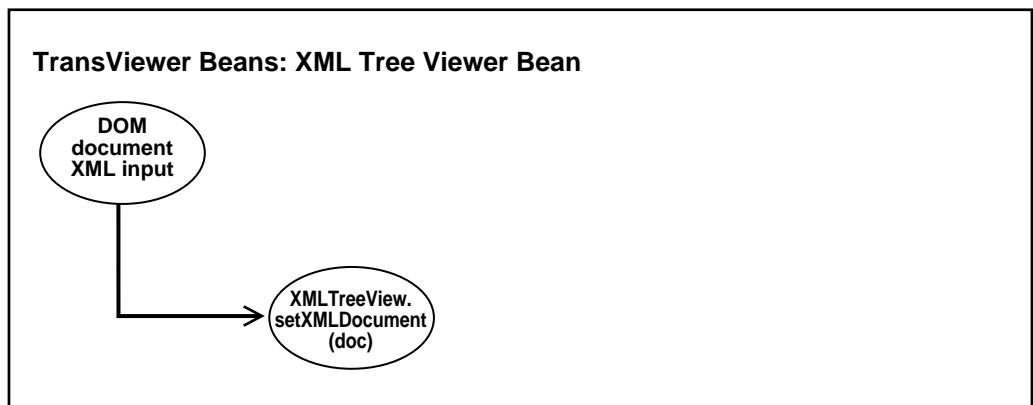
Figure 23–3 Treeviewer Bean in Action: Displaying an XML Document as a Tree

Figure 23-4 illustrates XML Treeviewer bean usage. A DOM XML document is input to `XMLTreeView.setXMLDocument(doc)` method. This associates the XML Treeviewer with the XML document. Available Treeviewer bean methods are:

- `getPreferredSize()` — Returns the `XMLTreeView` preferred size.
- `setXMLDocument(Document)` — Associates the `XMLTreeView` with an XML document.
- `updateUI()` — Forces the `XMLTreeView` to update/refresh the user interface.

Figure 23-4 XML Treeviewer Bean Usage



Using XMLSourceView Bean

`XMLSourceView` bean is a visual Java bean that displays an XML document. It improves the viewing of XML and XSL files by color-highlighting the XML/XSL syntax. It also offers an Edit mode. `XMLSourceView` bean easily integrates with `DOM Builder` bean. It allows for pre or post parsing visualization and validation against a specified DTD.

`XMLSourceView` bean recognizes the following XML token types:

- Tag
- Attribute Name
- Attribute Value
- Comment

- CDATA
- PCDATA
- PI Data
- PI Name
- NOTATION Symbol

Each token type has a foreground color and font. The default color/font settings can be changed by the user. This takes as input, an `org.w3c.dom.Document` object.

XMLSourceView Bean Usage

[Figure 23-6](#) shows the XMLSourceView bean usage. This is part of the `oracle.xml.srcviewer` API. A DOM document inputs `XMLSourceView.SetXMLDocument(Doc)`. The resulting DOM document is displayed. See "[Transviewer Bean Example 2: ViewSample.java](#)".

[Figure 23-5](#), "[XMLSourceView Bean in Action: Displaying an XML Document with Color Highlighting](#)" displays an XML document with tags shown in blue, tag content in black and attributes in red.

Figure 23–5 XMLSourceView Bean in Action: Displaying an XML Document with Color Highlighting

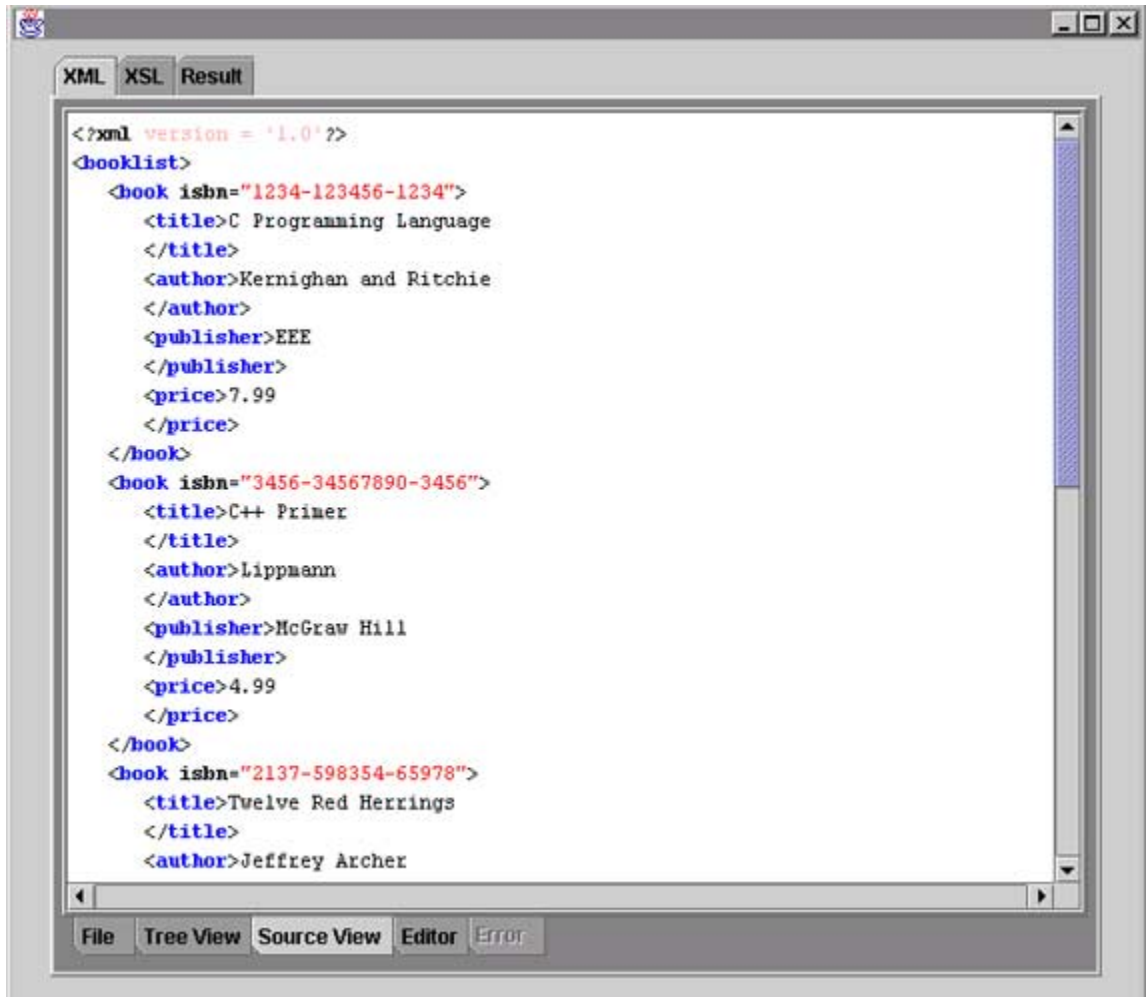


Figure 23–6 XMLSourceView Bean Usage

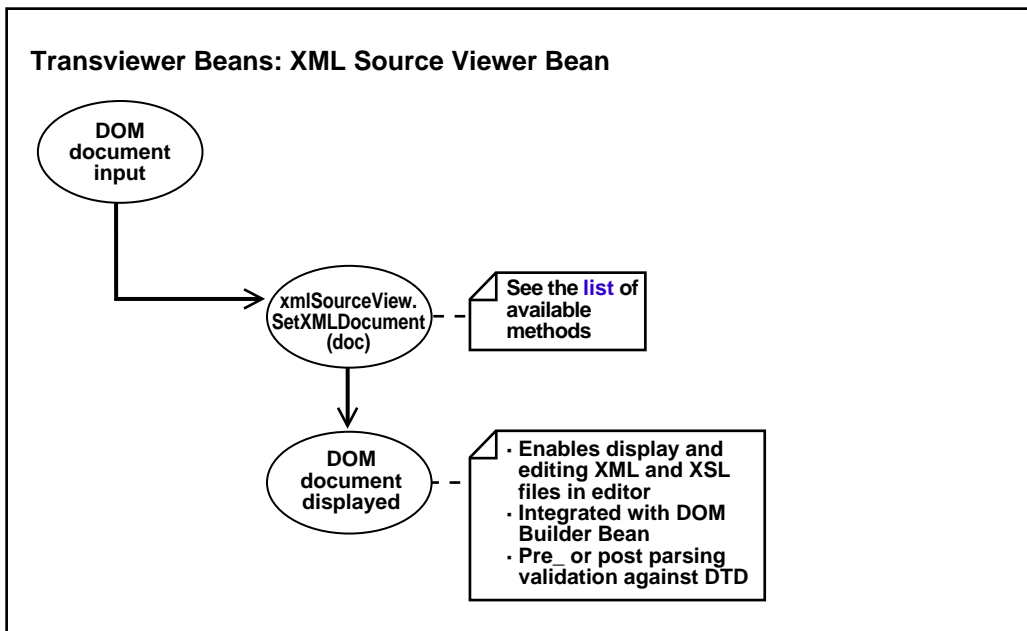


Table 23–3 lists the XMLSourceView methods.

Table 23–3 XMLSourceView Bean Methods

Method	Description
fontGet(AttributeSet)	Extracts and returns the font from a given attributeset.
fontSet(MutableAttributeSet, Font)	Sets the mutableattributeset font.
getAttributeNameFont()	Returns the Attribute Value font.
getAttributeNameForeground()	Returns the Attribute Name foreground color.
getAttributeValueFont()	Returns the Attribute Value font.
getAttributeValueForeground()	Returns the Attribute Value foreground color.
getBackground()	Returns the background color.
getCDATAFont()	Returns the CDATA font.

Table 23–3 XMLSourceView Bean Methods (Cont.)

Method	Description
getCDATAForeground()	Returns the CDATA foreground color.
getCommentDataFont()	Returns the Comment Data font.
getCommentDataForeground()	Returns the Comment Data foreground color.
getEditedText()	Returns the edited text.
getJTextPane()	Returns the viewer JTextPane component.
getMinimumSize()	Returns the XMLSourceView minimal size.
getNodeAtOffset(int)	Returns the XML node at a given offset.
getPCDATAFont()	Returns the PCDATA font.
getPCDATAForeground()	Returns the PCDATA foreground color.
getPIDataFont()	Returns the PI Data font.
getPIDataForeground()	Returns the PI Data foreground color.
getPINameFont()	Returns the PI Name font.
getPINameForeground()	Returns the PI Data foreground color.
getSymbolFont()	Returns the NOTATION Symbol font.
getSymbolForeground()	Returns the NOTATION Symbol foreground color.
getTagFont()	Returns the Tag font.
getTagForeground()	Returns the Tag foreground color.
getText()	Returns the XML document as a String.
isEditable()	Returns boolean to indicate whether this object is editable.
selectNodeAt(int)	Moves the cursor to XML Node at offset i.
setAttributeNameFont(Font)	Sets the Attribute Name font.
setAttributeNameForeground(Color)	Sets the Attribute Name foreground color.
setAttributeValueFont(Font)	Sets the Attribute Value font.
setAttributeValueForeground(Color)	Sets the Attribute Value foreground color.
setBackground(Color)	Sets the background color.
setCDATAFont(Font)	Sets the CDATA font.
setCDATAForeground(Color)	Sets the CDATA foreground color.

Table 23–3 XMLSourceView Bean Methods (Cont.)

Method	Description
setCommentDataFont(Font)	Sets the Comment font.
setCommentDataForeground(Color)	Sets the Comment foreground color.
setEditable(boolean)	Sets the specified boolean to indicate whether this object should be editable.
setPCDATAFont(Font)	Sets the PCDATA font.
setPCDATAForeground(Color)	Sets the PCDATA foreground color.
setPIDataFont(Font)	Sets the PI Data font.
setPIDataForeground(Color)	Sets the PI Data foreground color.
setPINameFont(Font)	Sets the PI Name font.
setPINameForeground(Color)	Sets the PI Name foreground color.
setSelectedNode(Node)	Sets the cursor position at the selected XML node.
setSymbolFont(Font)	Sets the NOTATION Symbol font.
setSymbolForeground(Color)	Sets the NOTATION Symbol foreground color.
setTagFont(Font)	Sets the Tag font.
setTagForeground(Color)	Sets the Tag foreground color.
setXMLDocument(Document)	Associates the XMLviewer with a XML document.

Using XMLTransformPanel Bean

XMLTransformPanel visual bean applies XSL transformations to XML documents. It visualizes the result and allows editing of input XML and XSL documents and files. XMLTransformPanel bean requires no programmatic input. It is a component that interacts directly with you and is not customizable.

XMLTransformPanel Bean Features

XMLTransformPanel bean has the following features:

- Imports and exports XML and XSL files from the file system, and XML, XSL, and HTML files from Oracle. With Oracle9i, XMLTransformPanel bean uses two-column CLOB tables. The first column stores the data name (file name) and the second stores the data text (file's data) in a CLOB. The bean lists all CLOB tables in your schema. When you click on a table, the bean lists its file names.

You can also create or delete tables, retrieve or add files to the tables. This can be useful for organizing your information. See [Figure 23-7](#), "[XSLTransformPanel Bean in Action: Showing CLOB Table and Data Names](#)"

Note: CLOB tables created by the XSL Transformer bean can be used by trigger-based stored procedures to mirror tables or views in the database *into HTML data* held in these CLOB tables. See "[XSL Transviewer Bean Scenario 1: Regenerating HTML — Underlying Data Changes](#)".

- Supports multiple database connections
- Creates XML from database result sets. This feature allows you to submit any SQL query to the database that you are currently connected. The bean converts the result set into XML and automatically loads this XML data into the bean's XML buffer for further processing.
- Edits XML and XSL data loaded into the bean.
- Applies XSL transformations to XML buffers and show the results. See With the bean, you can also export results to the file system or a CLOB in the database.

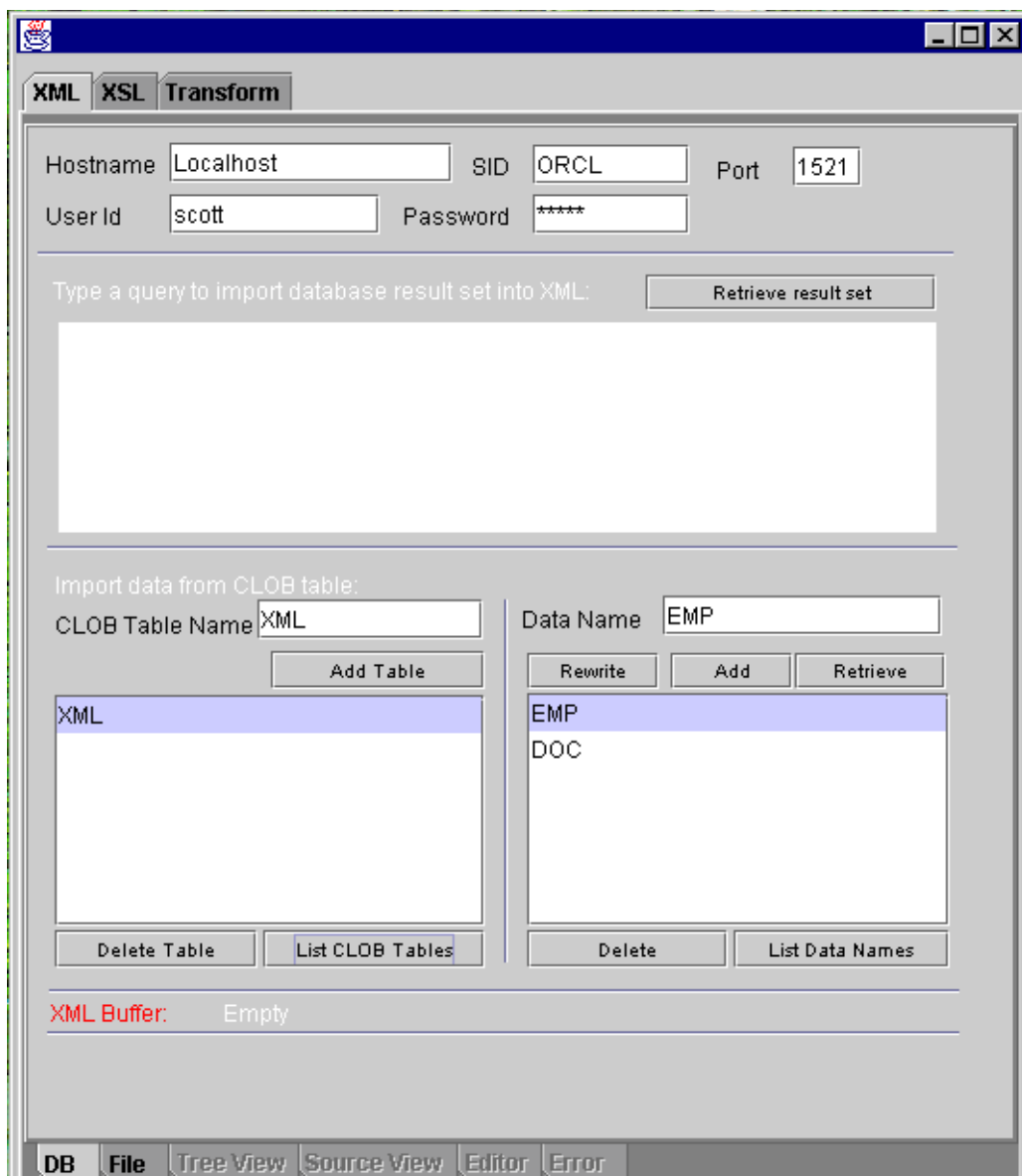
Transviewer Bean Application

The Transviewer bean is one application that illustrates the use of XMLTransformPanel bean. It can be used from a command line to perform the following actions:

- Edit and parse XML files
- Edit and apply XSL transformations
- Retrieve and save XML, XSL and result files in the file system or in Oracle

See: "[Transviewer Bean Example 3: XMLTransformPanelSample.java](#)" for an example of how to use XMLTransformPanel.

Figure 23–7 XSLTransformPanel Bean in Action: Showing CLOB Table and Data Names



Using DBViewer Bean

DBViewer bean can be used to display database queries on any XML document by applying XSL stylesheets and visualizing the resulting HTML in a scrollable swing panel. See:

- [Figure 23–8, "DBViewer Bean in Action: Entering a Database Query to Generate XML"](#)
- [Figure 23–9, "DBViewer Bean in Action: Viewing the XML Document After Transforming to HTML With XSL Stylesheet"](#)

DBViewer bean has the following three buffers:

- XML
- XSL
- Result buffer

DBViewer bean API allows the calling program to load or save buffers from various sources and apply stylesheet transformation to the XML buffer using the stylesheet in the XSL buffer. Results can be stored in the result buffer.

Showing Content

Content in the XML and XSL buffers can be shown as a source or tree structure. Content in the result buffer content can be rendered as HTML and also shown as a source or tree structure.

Loading and Saving the Buffers

The XML buffer can be loaded using a database query. All the buffers can be loaded from and files saved from the following:

- CLOB tables in Oracle
- File system

Therefore, control can also be used to move files between the file system and the user schema in the database.

Figure 23-8 DBViewer Bean in Action: Entering a Database Query to Generate XML

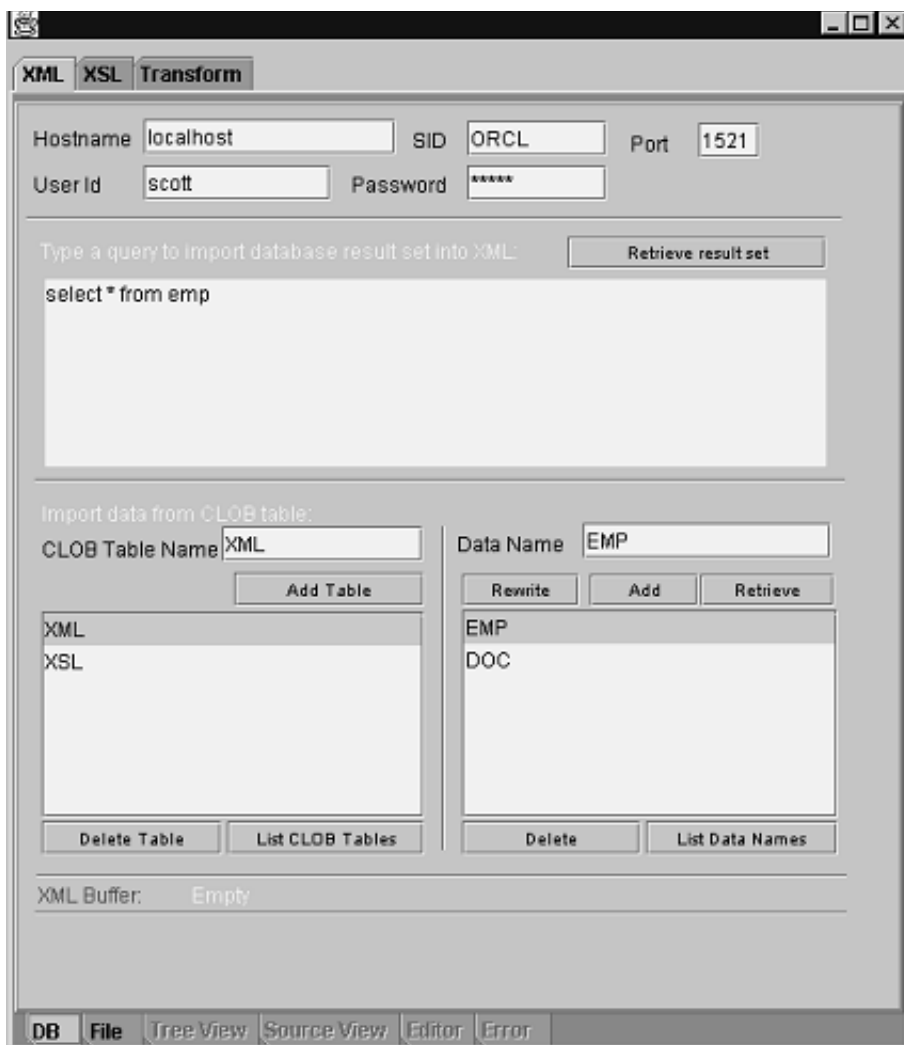
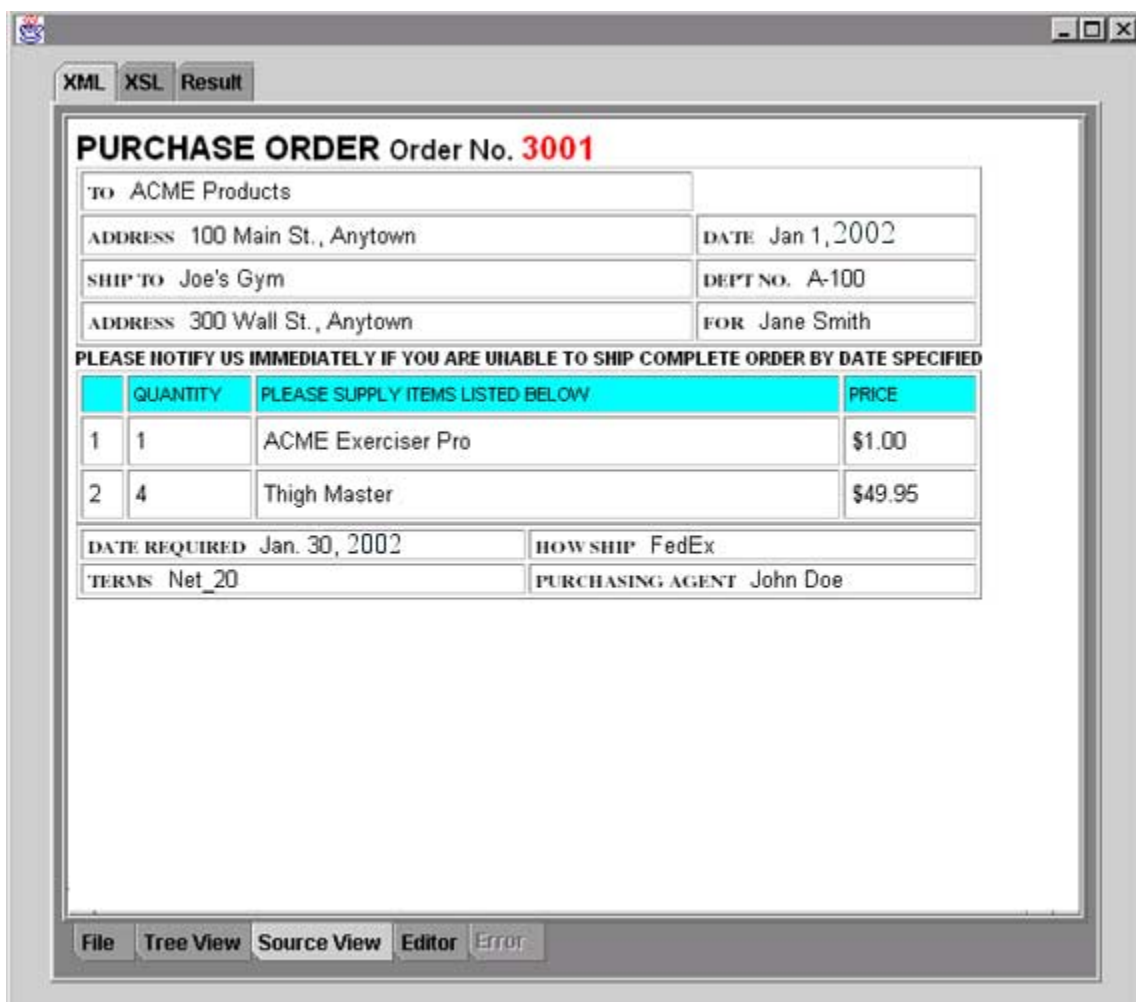


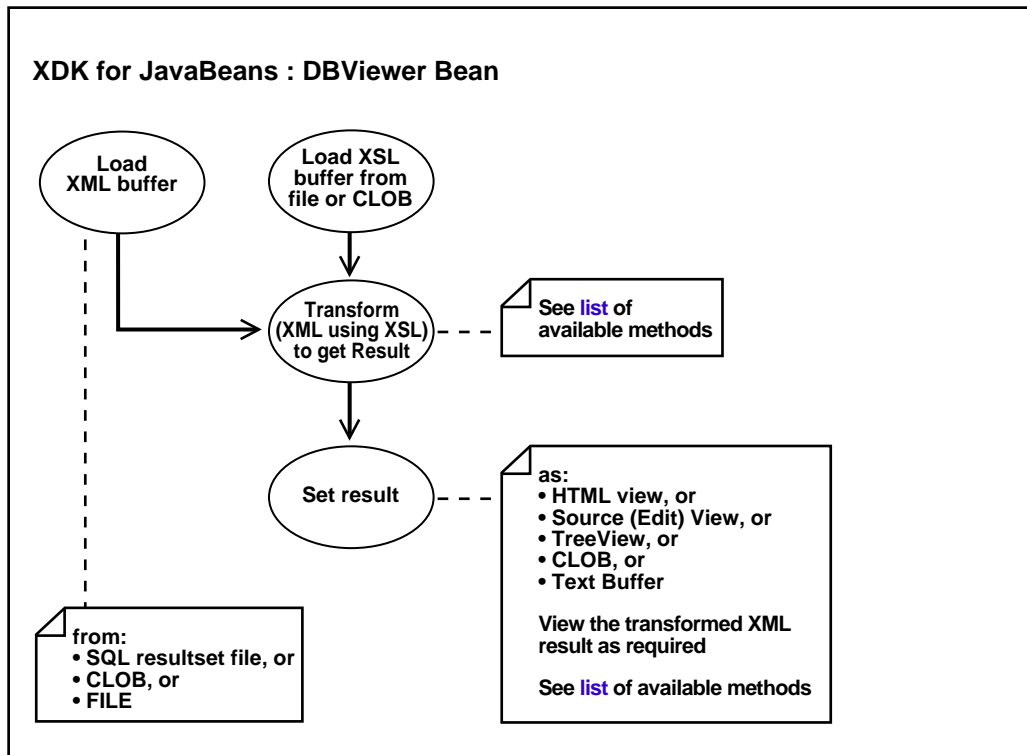
Figure 23–9 DBViewer Bean in Action: Viewing the XML Document After Transforming to HTML With XSL Stylesheet



DBViewer Bean Usage

Figure 23–10 illustrates DBViewer bean's usage.

Figure 23–10 DBViewer Bean Usage Diagram



DBViewer Bean Methods

Table 23–4 lists the DBViewer bean methods.

Table 23–4 DBViewer Bean Methods

Method	Description
DBViewer()	Constructs a new instance.
getHostname()	Gets database host name

Table 23–4 DBViewer Bean Methods(Cont.)

Method	Description
getInstancename()	Gets database instance name
getPassword()	Gets user password
getPort()	Gets database port number
getResBuffer()	Gets the content of the result buffer
getResCLOBFileName()	Gets result CLOB file name
getResCLOBTableName()	Gets result CLOB table name
getResFileName()	Gets Result file name
getUsername()	Gets user name
getXmlBuffer()	Gets the content of the XML buffer
getXmlCLOBFileName()	Gets XML CLOB file name
getXmlCLOBTableName()	Gets XML CLOB table name
getXmlFileName()	Gets XML file name
getXMLStringFromSQL(String)	Gets XML presentation of result set from SQL query
getXslBuffer()	Gets the content of the XSL buffer
getXslCLOBFileName()	Gets the XSL CLOB file name
getXslCLOBTableName()	Gets XSL CLOB table name
getXslFileName()	Gets XSL file name
loadResBuffer(String)	Loads the result buffer from file
loadResBuffer(String, String)	Loads the result buffer from CLOB file
loadResBufferFromClob()	Loads the result buffer from CLOB file
loadResBufferFromFile()	Loads the result buffer from file
loadXmlBuffer(String)	Loads the XML buffer from file
loadXmlBuffer(String, String)	Loads the XML buffer from CLOB file
loadXmlBufferFromClob()	Loads the XML buffer from CLOB file
loadXmlBufferFromFile()	Loads the XML buffer from file
loadXMLBufferFromSQL(String)	Loads the XML buffer from SQL result set
loadXslBuffer(String)	Loads the XSL buffer from file

Table 23–4 DBViewer Bean Methods(Cont.)

Method	Description
loadXslBuffer(String, String)	Loads the XSL buffer from CLOB file
loadXslBufferFromClob()	Loads the XSL buffer from CLOB file
loadXslBufferFromFile()	Loads the XSL buffer from file
parseResBuffer()	Parses the result buffer and refresh the tree view and source view
parseXmlBuffer()	Parses the XML buffer and refresh the tree view and source view
parseXslBuffer()	Parses the XSL buffer and refresh the tree view and source view
saveResBuffer(String)	Saves the result buffer to file
saveResBuffer(String, String)	Saves the result buffer to CLOB file
saveResBufferToClob()	Saves the result buffer to CLOB file
saveResBufferToFile()	Saves the result buffer to file
saveXmlBuffer(String)	Saves the XML buffer to file
saveXmlBuffer(String, String)	Saves the XML buffer to CLOB file
saveXmlBufferToClob()	Saves the XML buffer to CLOB file
saveXmlBufferToFile()	Saves the XML buffer to file
saveXslBuffer(String)	Saves the XSL buffer to file
saveXslBuffer(String, String)	Saves the XSL buffer to CLOB file
saveXslBufferToClob()	Saves the XSL buffer to CLOB file
saveXslBufferToFile()	Saves the XSL buffer to file
setHostname(String)	Sets database host name
setInstancename(String)	Sets database instance name
setPassword(String)	Sets user password
setPort(String)	Sets database port number
setResBuffer(String)	Sets new text in the result buffer
setResCLOBFileName(String)	Sets Result CLOB file name
setResCLOBTableName(String)	Sets Result CLOB table name

Table 23–4 DBViewer Bean Methods(Cont.)

Method	Description
setResFileName(String)	Sets Result file name
setResHtmlView(boolean)	Shows the result buffer as rendered HTML
setResSourceEditView(boolean)	Shows the result buffer as XML source and enter edit mode
setResSourceView(boolean)	Shows the result buffer as XML source
setResTreeView(boolean)	Shows the result buffer as XML tree view
setUsername(String)	Sets user name
setXmlBuffer(String)	Sets new text in the XML buffer
setXmlCLOBFileName(String)	Sets XML CLOB table name
setXmlCLOBTableName(String)	Sets XML CLOB table name
setXmlFileName(String)	Sets XML file name
setXmlSourceEditView(boolean)	Shows the XML buffer as XML source and enter edit mode
setXmlSourceView(boolean)	Shows the XML buffer as XML source
setXmlTreeView(boolean)	Shows the XML buffer as tree
setXslBuffer(String)	Sets new text in the XSL buffer
setXslCLOBFileName(String)	Sets XSL CLOB file name
setXslCLOBTableName(String)	Sets XSL CLOB table name
setXslFileName(String)	Sets XSL file name
setXslSourceEditView(boolean)	Shows the XSL buffer as XML source and enter edit mode
setXslSourceView(boolean)	Shows the XSL buffer as XML source
setXslTreeView(boolean)	Shows the XSL buffer as tree
transformToDoc()	Transforms the content of the XML buffer by applying the stylesheet from the XSL buffer.
transformToRes()	Applies the stylesheet transformation from the XSL buffer to the XML in the XML buffer and stores the result into the result buffer
transformToString()	Transforms the content of the XML buffer by applying the stylesheet from the XSL buffer.

Using DBAccess Bean

DBAccess bean maintains CLOB tables that can hold multiple XML and text documents. Each table is created using the following statement:

```
CREATE TABLE tablename FILENAME CHAR( 16) UNIQUE, FILEDATA CLOB) LOB(FILEDATA)  
STORE AS (DISABLE STORAGE IN ROW)
```

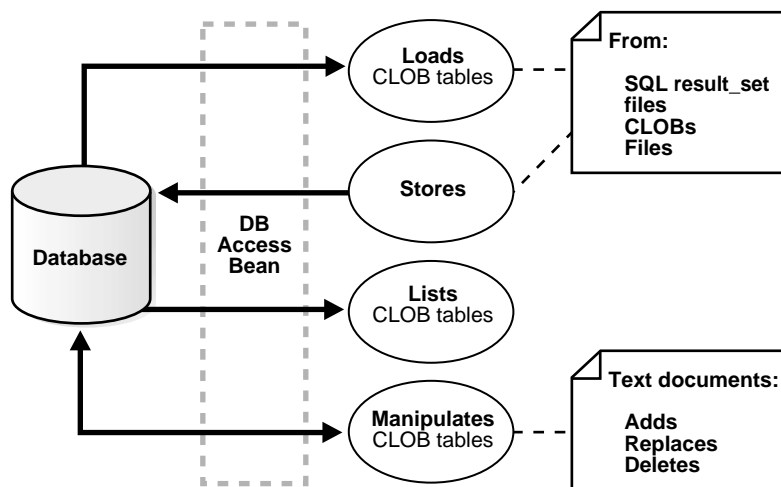
Each XML (or text) document is stored as a row in the table. The FILENAME field holds a unique string used as a key to retrieve, update, or delete the row. Document text is stored in the FILEDATA field. This is a CLOB object. CLOB tables are automatically maintained by the Transviewer bean. The CLOB tables maintained by DBAccess bean can be later used by the Transviewer bean. DBAccess bean does the following tasks:

- Creates and deletes CLOB tables
- Lists a CLOB table's contents
- Adds, replaces, or deletes text documents in the CLOB tables

DBAccess Bean Usage

[Figure 23-11](#) illustrates the DBAccess bean usage. It shows how DBAccess bean maintains, and manipulates XML documents stored in CLOBs.

Figure 23–11 DBAccess Bean Usage Diagram



DBAccess Bean Methods

Table 23–5 lists the DBAccess bean methods.

Table 23–5 DBAccess Bean Methods

Method	Description
createXMLTable(Connection, String)	Creates XML table
deleteXMLName(Connection, String, String)	Deletes text file from XML table
dropXMLTable(Connection, String)	Deletes XML table
getNameSize()	Returns the size of the field where the filename is kept.
getXMLData(Connection, String, String)	Retrieve text file from XML table
getXMLNames(Connection, String)	Returns all file names in XML table
getXMLTableNames(Connection, String)	Gets all XML tables with names starting with a given string
insertXMLData(Connection, String, String, String)	Inserts text file as a row in XML table

Table 23–5 DBAccess Bean Methods (Cont.)

Method	Description
isXMLTable(Connection, String)	Checks if the table is XML table.
replaceXMLData(Connection, String, String, String)	Replaces text file as a row in XML table
xmlTableExists(Connection, String)	Checks if XML table exists.

Running the Transviewer Bean Samples

The XDK for Java Transviewer bean sample/ directory contains sample Transviewer bean applications that illustrate how to use Oracle Transviewer beans. Oracle Transviewer beans toolset contains DOMBuilder, XMLSourceView, XMLTreeView, XSLTransformer, XMLTransformPanel, DBViewer, and DBAccess beans.

[Table 23–6](#) lists the sample files in sample/.

Table 23–6 Transviewer Bean Sample Files in sample/

File Name	Description
booklist.xml	Sample XML file used by Example 1, 2, or 3
doc.xml	Sample XML file used by Example 1, 2, or 3
doc.html	Sample HTML file used by Examples 1, 2, or 3
doc.xsl	Sample input XSL file used by Examples 1, 2, or 3. doc.xsl is used by XSLTransformer.
emtable.xsl	Sample input XSL file used by Examples 1, 2, or 3
tohtml.xsl	Sample input XSL file used by Examples 1, 2, or 3. Transforms booklist.xml.
AsyncTransformSample.java See " Transviewer Bean Example 1: AsyncTransformSample.java ".	Sample nonvisual application using XSLTransformer bean and DOMBuilder bean. It applies the XSLT stylesheet specified in doc.xsl on all *.xml files from the current directory. The results are in the files with extension.log.

Table 23–6 *Transviewer Bean Sample Files in sample/*

File Name	Description
ViewSample.java See "Transviewer Bean Example 2: ViewSample.java".	Sample visual application that uses XMLSourceView and XMLTreeView beans. It visualizes XML document files.
XMLTransformPanelSample.java See "Transviewer Bean Example 3: XMLTransformPanelSample.java".	A visual application that uses XMLTransformPanel bean. This bean uses all four beans from above. It applies XSL transformations on XML documents and shows the result. Visualizes and allows editing of XML and XSL input files.
DBViewSample See: <ul style="list-style-type: none"> ▪ "Transviewer Bean Example 4a: DBViewer Bean — DBViewClaims.java" ▪ "Transviewer Bean Example 4b: DBViewer Bean — DBViewFrame.java" ▪ "Transviewer Bean Example 4c: DBViewer Bean — DBViewSample.java" 	A sample visual application that uses DBViewer bean to implement simple insurance claim handling application.

Installing the Transviewer Bean Samples

The Transviewer beans require as a minimum JDK 1.1.6, and can be used with any version of JDK 1.2.

1. Download and install the following components used by the Transviewer beans:
 - Oracle JDBC Driver for thin client (jar file classes111.zip)
 - Oracle XML SQL Utility (jar file oraclexmlsql.jar)

After installing this components, include classes111.zip and oraclexmlsql.jar in your classpath.
2. The beans and the samples use swing 1.1. If you use jdk1.2, go to step 3. If you use jdk1.1, you will need to download Swing 1.1 from Sun. After downloading Swing, add swingall.jar to your classpath.
3. Change JDKPATH in `Makefile` to point to your JDK path. In addition, on Windows NT, change the file separator as stated in the `Makefile`.
4. If you are not using the default database with a scott/tiger account, change USERID and PASSWORD in the `Makefile` to run Sample4
5. Run "make" to generate .class files.

6. Run the sample programs using commands:
 - **gmake** sample1
 - **gmake** sample2
 - **gmake** sample3
 - **gmake** sample4
7. Visualize the results in .log files using the ViewSample.
8. Use the XSLT document from './tohtml.xml' to transform the XML document from './booklist.xml'.

A few .xml files are provided as test cases. An XSL stylesheet 'doc.xml' is used by XSLTransformer.

Note: sample1 runs the XMLTransViewer program so that you can import and export XML files from Oracle, keep your XSL transformation files in Oracle, and apply stylesheets to XML interactively.

Using Database Connectivity

To use the database connectivity feature in this program, you must know the following:

- Network name of the computer where Oracle or Oracle Application Server runs
- Port (usually 1521)
- Name of the oracle instance (usually orcl)

You also need an account with CREATE TABLE privilege.

You can try the default account scott with password tiger if it still enabled on your Oracle system.

Running Makefile

The following is the makefile script:

```
# Makefile for sample java files

.SUFFIXES : .java .class
```



```

CLASSES = ViewSample.class AsyncTransformSample.class
XMLTransformPanelSample.class

# Change it to the appropriate separator based on the OS
PATHSEP= :

# Change this path to your JDK location. If you use JDK 1.1, you will need
# to download also Swing 1.1 and add swingall.jar to your classpath.
# You do not need to do this for JDK 1.2 since Swing is part of JDK 1.2
JDKPATH = /usr/local/packages/jdk1.2

# Make sure that the following product jar/zip files are in the classpath:
# - Oracle JDBC driver for thin client (file classes111.zip)
# - Oracle XML SQL Utility (file oraclexmlsql.jar)
# You can download this products from technet.us.oracle.com

#
CLASSPATH
:=$(CLASSPATH)$(PATHSEP)../lib/xmlparserv2.jar$(PATHSEP)../lib/xmlcomp.jar$(PATH
SEP)../lib/jdev-rt.zip$(PATHSEP).$(PATHSEP)
%.class: %.java
$(JDKPATH)/bin/javac -classpath "$(CLASSPATH)" $<

# make all class files
all: $(CLASSES)

sample1: XMLTransformPanelSample.class
$(JDKPATH)/bin/java -classpath "$(CLASSPATH)" XMLTransformPanelSample
sample2: ViewSample.class
$(JDKPATH)/bin/java -classpath "$(CLASSPATH)" ViewSample
sample3: AsyncTransformSample.class
$(JDKPATH)/bin/java -classpath "$(CLASSPATH)" AsyncTransformSample

```

Transviewer Bean Example 1: AsyncTransformSample.java

This example shows you how to use DOMBuilder and the XSLTransformer beans to asynchronously transform multiple XML files.

```

import java.net.URL;
import java.net.MalformedURLException;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;

```

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.util.Vector;

import org.w3c.dom.DocumentFragment;
import org.w3c.dom.DOMException;

import oracle.xml.async.DOMBuilder;
import oracle.xml.async.DOMBuilderEvent;
import oracle.xml.async.DOMBuilderListener;
import oracle.xml.async.DOMBuilderErrorEvent;
import oracle.xml.async.DOMBuilderErrorListener;
import oracle.xml.async.XSLTransformer;
import oracle.xml.async.XSLTransformerEvent;
import oracle.xml.async.XSLTransformerListener;
import oracle.xml.async.XSLTransformerErrorEvent;
import oracle.xml.async.XSLTransformerErrorListener;
import oracle.xml.async.ResourceManager;
import oracle.xml.parser.v2.DOMParser;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.XSLStylesheet;
import oracle.xml.parser.v2.*;

public class AsyncTransformSample
{
    /**
     * uses DOMBuilder bean
     */
    void runDOMBuilders ()
    {
        rm = new ResourceManager (numXMLDocs);

        for (int i = 0; i < numXMLDocs; i++)
        {
            rm.getResource();

            try
            {
                DOMBuilder builder = new DOMBuilder(i);

                URL xmlURL = createURL(basedir + "/" +
                                     (String)xmlfiles.elementAt(i));

                if (xmlURL == null)
                    exitWithError("File " + (String)xmlfiles.elementAt(i) +
```

```

        " not found");

builder.setPreserveWhitespace(true);
builder.setBaseURL (createUrl(basedir + "/"));
builder.addDOMBuilderListener (new DOMBuilderListener() {
    public void domBuilderStarted(DOMBuilderEvent p0) {}
    public void domBuilderError(DOMBuilderEvent p0) {}
    public synchronized void domBuilderOver(DOMBuilderEvent p0)
    {
        DOMBuilder bld = (DOMBuilder)p0.getSource();
        runXSLTransformer (bld.getDocument(), bld.getId());
    }
});
builder.addDOMBuilderErrorListener (new DOMBuilderErrorListener() {
    public void domBuilderErrorCalled(DOMBuilderErrorEvent p0)
    {
        int id = ((DOMBuilder)p0.getSource()).getId();
        exitWithError("Error occurred while parsing " +
            xmlfiles.elementAt(id) + ": " +
            p0.getException().getMessage());
    }
});
builder.parse (xmlURL);

    System.err.println("Parsing file " + xmlfiles.elementAt(i));
}
catch (Exception e)
{
    exitWithError("Error occurred while parsing " +
        (String)xmlfiles.elementAt(i) + ": " +
        e.getMessage());
}
}
}

/**
 * uses XSLTransformer bean
 */
void runXSLTransformer (XMLDocument xml, int id)
{
    try
    {
        XSLTransformer processor = new XSLTransformer (id);
        XSLStylesheet xsl      = new XSLStylesheet (xsldoc, xslURL);
    }
}

```

```
processor.showWarnings (true);
processor.setErrorStream (errors);
processor.addXSLTransformerListener (new XSLTransformerListener() {
    public void xslTransformerStarted (XSLTransformerEvent p0) {}
    public void xslTransformerError(XSLTransformerEvent p0) {}
    public void xslTransformerOver (XSLTransformerEvent p0)
    {
        XSLTransformer trans = (XSLTransformer)p0.getSource();
        saveResult (trans.getResult(), trans.getId());
    }
});
processor.addXSLTransformerErrorListener (new XSLTransformerErrorListener() {
public void xslTransformerErrorCalled(XSLTransformerErrorEvent p0)
    {
        int i = ((XSLTransformer)p0.getSource()).getId();
        exitWithError("Error occurred while processing " +
            xmlfiles.elementAt(i) + ": " +
            p0.getException().getMessage());
    }
});
processor.processXSL (xsl, xml);
// transform xml document
}
catch (Exception e)
{
    exitWithError("Error occurred while processing " + xslFile + ": " +
        e.getMessage());
}
}

void saveResult (DocumentFragment result, int id)
{
    System.err.println("Transforming '" + xmlfiles.elementAt(id) +
        "' to '" + xmlfiles.elementAt(id) + ".log'" +
        " applying '" + xslFile);

    try
    {
        File resultFile = new File((String)xmlfiles.elementAt(id) + ".log");

        ((XMLNode)result).print(new FileOutputStream(resultFile));
    }
    catch (Exception e)
    {
        exitWithError("Error occurred while generating output : " +
```

```
                e.getMessage());
            }

            rm.releaseResource();
        }

void makeXSLDocument ()
{
    System.err.println ("Parsing file " + xslFile);
    try
    {
        DOMParser parser = new DOMParser();
        parser.setPreserveWhitespace (true);
        xslURL = createURL (xslFile);
        parser.parse (xslURL);
        xsldoc = parser.getDocument();
    }
    catch (Exception e)
    {
        exitWithError("Error occurred while parsing " + xslFile + ": " +
            e.getMessage());
    }
}

private URL createURL(String fileName) throws Exception
{
    URL url = null;

    try
    {
        url = new URL(fileName);
    }
    catch (MalformedURLException ex)
    {
        File f = new File(fileName);

        try
        {
            String path = f.getAbsolutePath();
            // This is a bunch of weird code that is required to
            // make a valid URL on the Windows platform, due
            // to inconsistencies in what getAbsolutePath returns.
            String fs = System.getProperty("file.separator");
            if (fs.length() == 1)
            {
```

```
        char sep = fs.charAt(0);
        if (sep != '/')
            path = path.replace(sep, '/');
        if (path.charAt(0) != '/')
            path = '/' + path;
    }
    path = "file://" + path;
    url = new URL(path);
}
catch (MalformedURLException e)
{
    exitWithError("Cannot create url for: " + fileName);
}
}

return url;
}

boolean init () throws Exception
{
    File    directory = new File (basedir);
    String[] dirfiles = directory.list();
    for (int j = 0; j < dirfiles.length; j++)
    {
        String dirfile = dirfiles[j];

        if (!dirfile.endsWith(".xml"))
            continue;

        xmlfiles.addElement(dirfile);
    }

    if (xmlfiles.isEmpty()) {
        System.out.println("No files in directory were selected for processing");
        return false;
    }
    numXMLDocs = xmlfiles.size();

    return true;
}

private void exitWithError(String msg)
{
    PrintWriter errs = new PrintWriter(errors);
    errs.println(msg);
}
```

```
        errs.flush();
        System.exit(1);
    }

    void asyncTransform () throws Exception
    {
        System.err.println (numXMLDocs +
            " XML documents will be transformed" +
            " using XSLT stylesheet specified in " + xslFile +
            " with " + numXMLDocs + " threads");

        makeXSLDocument ();
        runDOMBuilders ();

        // wait for the last request to complete
        while (rm.activeFound())
            Thread.sleep(100);
    }

    String      basedir = new String (".");
    OutputStream errors = System.err;

    Vector xmlfiles = new Vector();
    int      numXMLDocs = 1;

    String      xslFile = new String ("doc.xsl");
    URL         xslURL;
    XMLDocument xsldoc;

    private ResourceManager rm;

    /**
     *   main
     */
    public static void main (String args[])
    {
        AsyncTransformSample inst = new AsyncTransformSample();

        try
        {
            if (!inst.init())
                System.exit(0);

            inst.asyncTransform ();
        }
    }
}
```

```
        catch (Exception e)
        {
            e.printStackTrace();
        }

        System.exit(0);
    }
}
```

Transviewer Bean Example 2: ViewSample.java

This example shows you how to use XMLSourceView and XMLTreeView beans to visually represent XML files.

```
import java.awt.*;
import oracle.xml.srcviewer.*;
import oracle.xml.treeviewer.*;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.*;
import org.w3c.dom.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class ViewSample
{
    public static void main(String[] args)
    {
        String fileName = new String ("booklist.xml");
        if (args.length > 0) {
            fileName = args[0];
        }

        JFrame      frame      = setFrame ("XMLViewer");
        XMLDocument xmlDocument = getXMLDocumentFromFile (fileName);
        XMLSourceView xmlSourceView = setXMLSourceView (xmlDocument);
        XMLTreeView  xmlTreeView  = setXMLTreeView (xmlDocument);
        JTabbedPane jtbPane      = new JTabbedPane ();

        jtbPane.addTab ("Source", null, xmlSourceView, "XML document source view");
        jtbPane.addTab ("Tree", null, xmlTreeView, "XML document tree view");
    }
}
```



```
        jtbpane.setPreferredSize (new Dimension(400,300));
        frame.getContentPane().add (jtbpane);

        frame.setTitle      (fileName);
        frame.setJMenuBar  (setMenuBar());
        frame.setVisible   (true);
    }

    static JFrame setFrame (String title)
    {
        JFrame frame = new JFrame (title);
        //Center the window
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation ((screenSize.width - frameSize.width)/2,
                           (screenSize.height - frameSize.height)/2);
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        frame.getContentPane().setLayout (new BorderLayout());
        frame.setSize(new Dimension(400, 300));
        frame.setVisible (false);
        frame.setTitle (title);

        return frame;
    }

    static JMenuBar setMenuBar ()
    {
        JMenuBar menuBar = new JMenuBar();
        JMenu menu = new JMenu ("Exit");
        menu.addMenuListener ( new MenuListener () {
            public void menuSelected (MenuEvent ev) { System.exit(0); }
            public void menuDeselected (MenuEvent ev) {}
            public void menuCanceled (MenuEvent ev) {}
        });
        menuBar.add (menu);
    }
}
```

```
        return menuBar;
    }

    /**
     * creates XMLSourceView object
     */
    static XMLSourceView setXMLSourceView(XMLDocument xmlDoc)
    {
        XMLSourceView xmlView = new XMLSourceView();

        xmlView.setXMLDocument(xmlDoc);
        xmlView.setBackground(Color.yellow);
        xmlView.setEditable(true);
        return xmlView;
    }

    /**
     * creates XMLTreeView object
     */
    static XMLTreeView setXMLTreeView(XMLDocument xmlDoc)
    {
        XMLTreeView xmlView = new XMLTreeView();

        xmlView.setXMLDocument(xmlDoc);
        xmlView.setBackground(Color.yellow);
        return xmlView;
    }

    static XMLDocument getXMLDocumentFromFile (String fileName)
    {
        XMLDocument doc = null;

        try {
            DOMParser parser = new DOMParser();
            try {
                String dir= "" ;
                FileInputStream in = new FileInputStream(fileName);
                parser.setPreserveWhitespace(false);
                parser.setBaseURL(createURL(dir));
                parser.parse(in);
                in.close();
            } catch (Exception ex) {
                ex.printStackTrace();
                System.exit(0);
            }
        }
    }
}
```

```
doc = (XMLDocument)parser.getDocument();

try {
    doc.print(System.out);
} catch (Exception ie) {
    ie.printStackTrace();
    System.exit(0);
}

}

catch (Exception e) {
    e.printStackTrace();
}

return doc;
}

static URL createURL(String fileName)
{
    URL url = null;
    try
    {
        url = new URL(fileName);
    }
    catch (MalformedURLException ex)
    {
        File f = new File(fileName);
        try
        {
            {
                String path = f.getAbsolutePath();
                String fs = System.getProperty("file.separator");
                if (fs.length() == 1)
                {
                    char sep = fs.charAt(0);
                    if (sep != '/')
                        path = path.replace(sep, '/');
                    if (path.charAt(0) != '/')
                        path = '/' + path;
                }
                path = "file://" + path;
                url = new URL(path);
            }
        }
        catch (MalformedURLException e)
        {
            System.out.println("Cannot create url for: " + fileName);
            System.exit(0);
        }
    }
}
```

```
        }  
    }  
    return url;  
}  
}
```

Transviewer Bean Example 3: XMLTransformPanelSample.java

This example is an interactive application that uses XMLTransformPanel bean to do the following:

- Generate XML from database queries
- Transform the XML using XSL stylesheets
- View the results
- Store the results in CLOB tables in the database

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
import oracle.xml.transviewer.XMLTransformPanel;  
  
public class XMLTransformPanelSample  
{  
    XMLTransformPanel transformPanel = new XMLTransformPanel();  
  
    /**  
     * Adjust frame size and add transformPanel to it.  
     */  
    public XMLTransformPanelSample ()  
    {  
        Frame frame = new JFrame();  
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();  
        frame.setSize(510,550);  
        transformPanel.setPreferredSize(new Dimension(510,550));  
        Dimension frameSize = frame.getSize();  
  
        if (frameSize.height > screenSize.height) {  
            frameSize.height = screenSize.height;  
        }  
        if (frameSize.width > screenSize.width) {  
            frameSize.width = screenSize.width;  
        }  
    }  
}
```

```

        frame.setLocation ((screenSize.width - frameSize.width)/2,
                           (screenSize.height - frameSize.height)/2);
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) { System.exit(0); }
        });
        frame.setVisible(true);

        ((JFrame)frame).getContentPane().add (transformPanel);
        frame.pack();
    }

    /**
     * main(). Only creates XMLTransformPanelSample object.
     */
    public static void main (String[] args)
    {
        new XMLTransformPanelSample ();
    }
}

```

Transviewer Bean Example 4a: DBViewer Bean — DBViewClaims.java

This is an interactive example which lets you input the name or policy of an insurance claim. The appropriate claim is loaded as an XML buffer from the result set of an XML query. An XSL stylesheet is loaded from the file system. The DBViewer bean transforms the XML buffer using the XSL stylesheet to HTML. This HTML output can then be viewed.

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import oracle.jdeveloper.layout.*;
import oracle.xml.dbviewer.*;

public class DBViewClaims extends JPanel {
    DBViewer dbPanel= new DBViewer();
    JButton searchButton = new JButton();
    XYLayout xYLayout1 = new XYLayout();
    JLabel titleLabel = new JLabel();
    JLabel nameLabel = new JLabel();
    JLabel policyLabel = new JLabel();
    JTextField nameTF = new JTextField();
    JTextField policyTF = new JTextField();
    JButton viewXMLButton = new JButton();
    JButton viewXSLButton = new JButton();
}

```

```
        JButton viewHTMLButton = new JButton();
    public DBViewClaims() {
        super();
        try {
            jbInit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
    private void jbInit() throws Exception {
        setBackground(SystemColor.controlLtHighlight);
        this.setLayout(xYLayout1);
        searchButton.setText("searchButton");
        searchButton.setLabel("Search");
        xYLayout1.setHeight(464);
        xYLayout1.setWidth(586);
        titleLabel.setText("List of Claims");
        titleLabel.setHorizontalAlignment(SwingConstants.CENTER);
        titleLabel.setBackground(new Color(192, 192, 255));
        titleLabel.setFont(new Font("Dialog", 1, 16));
        nameLabel.setText("Last Name");
        policyLabel.setText("Policy:");
        viewXMLButton.setText("viewXMLButton");
        viewXMLButton.setLabel("view XML");
        viewXMLButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                viewXMLButton_actionPerformed(e);
            }
        });
        viewXSLButton.setText("viewXSLButton");
        viewXSLButton.setLabel("view XSL");
        viewXSLButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                viewXSLButton_actionPerformed(e);
            }
        });
        viewHTMLButton.setText("viewHTMLButton");
        viewHTMLButton.setLabel("view HTML");
        viewHTMLButton.addActionListener(new java.awt.event.ActionListener() {

            public void actionPerformed(ActionEvent e) {
                viewHTMLButton_actionPerformed(e);
            }
        });
    }
```

```

searchButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        searchButton_actionPerformed(e);
    }
});

this.add(dbPanel, new XYConstraints(16, 55, 552, 302));
this.add(searchButton, new XYConstraints(413, 415, 154, 29));
this.add(titleLabel, new XYConstraints(79, 10, 413, 31));
this.add(nameLabel, new XYConstraints(333, 373, 72, -1));
this.add(policyLabel, new XYConstraints(334, 395, 59, -1));
this.add(nameTF, new XYConstraints(413, 368, 155, -1));
this.add(policyTF, new XYConstraints(413, 391, 156, -1));
this.add(viewXMLButton, new XYConstraints(19, 359, 94, 29));
this.add(viewXSLButton, new XYConstraints(19, 390, 94, 29));
this.add(viewHTMLButton, new XYConstraints(19, 421, 94, 29));
updateUI();
}
void searchButton_actionPerformed(ActionEvent e) {
    String sqlText="select * from s_claim c ";
    try {
        if (!nameTF.getText().equals("")) {
            sqlText=sqlText+" where c.claimpolicy.primaryinsured.lastname="+
                ""+nameTF.getText()+" ";
        } else if (!policyTF.getText().equals("")) {
            sqlText=sqlText+" where c.claimpolicy.policyid="+
                policyTF.getText();
        }
        dbPanel.setUsername("scott");
        dbPanel.setPassword("tiger");
        dbPanel.setInstancename("orcl");
        dbPanel.setHostname("localhost");
        dbPanel.setPort("1521");
        dbPanel.loadXMLBufferFromSQL(sqlText);
        dbPanel.loadXslBuffer("xslfiles","CLAIM.XSL");
        dbPanel.transformToRes();
        dbPanel.setResHtmlView(true);
    } catch (Exception e1) {
        System.out.println(e1);
    }
}
void viewXMLButton_actionPerformed(ActionEvent e) {
    dbPanel.setXmlSourceEditView(true);
}
}

```

```
void viewXSLButton_actionPerformed(ActionEvent e) {
    dbPanel.setXslSourceEditView(true);
}
void viewHTMLButton_actionPerformed(ActionEvent e) {
    dbPanel.setResHtmlView(true);
}
}
```

Transviewer Bean Example 4b: DBViewer Bean — DBViewFrame.java

This example provides a frame with a menu bar to access the DBView Claims functionality. Claims can then be loaded and displayed in HTML.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import oracle.jdeveloper.layout.*;

public class DBViewFrame extends JFrame {
    JMenuBar menuBar1 = new JMenuBar();
    JMenu menuFile = new JMenu();
    JMenuItem menuFileExit = new JMenuItem();
    JMenuItem menuListCustomerClaims = new JMenuItem();

    public DBViewFrame() {
        super();
        try {
            jbInit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        this.getContentPane().setLayout(new GridLayout(1,1));
        this.setSize(new Dimension(600, 550));
        menuFile.setText("File");
        menuFileExit.setText("Exit");
        menuListCustomerClaims.setText("List Claims");
        menuFileExit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                fileExit_ActionPerformed(e);
            }
        });
        menuListCustomerClaims.addActionListener(new ActionListener() {
```



```

        public void actionPerformed(ActionEvent e) {
            ListCustomerClaims_ActionPerformed(e);

        }
    });
    menuFile.add(menuFileExit);
    menuFile.add(menuListCustomerClaims);
    menuBar1.add(menuFile);
    this.setJMenuBar(menuBar1);
    this.setBackground(SystemColor.controlLtHighlight);
}
void fileExit_ActionPerformed(ActionEvent e) {
    System.exit(0);
}
void ListCustomerClaims_ActionPerformed(ActionEvent e) {
    this.getContentPane().removeAll();
    this.getContentPane().add(new DBViewClaims());
    this.getContentPane().paintAll(this.getGraphics());
}
}
}

```

Transviewer Bean Example 4c: DBViewer Bean — DBViewSample.java

This example simply provides a main function which instantiates DBViewFrame, giving it a specific look and feel.

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class DBViewSample {
    public DBViewSample() {
        DBViewFrame frame = new DBViewFrame();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        new DBViewSample();
    }
}

```


Part VIII

XDK for C

Part VIII describes how to access and use XML Developer's Kit (XDK) for C. It contains the following chapters:

- [Chapter 24, "Using XML Parser for C"](#)
- [Chapter 25, "Using XML Schema Processor for C"](#)

Using XML Parser for C

This chapter contains the following sections:

- [Accessing XML Parser for C](#)
- [XML Parser for C Features](#)
- [XML Parser for C Usage](#)
- [XML Parser for C, XSLT \(DOM Interface\) Usage](#)
- [XML Parser for C, Default Behavior](#)
- [DOM and SAX APIs](#)
- [Invoking XML Parser for C](#)
- [Using the Sample Files Included with Your Software](#)
- [Running the XML Parser for C Sample Programs](#)

Accessing XML Parser for C

XML Parser for C is provided with Oracle and Oracle Application Server. It is also available for download from the OTN site: <http://otn.oracle.com/tech/xml>

It is located in `$ORACLE_HOME/xdk/c/parser`.

XML Parser for C Features

`readme.html` in the root directory of the software archive contains release specific information including bug fixes and API additions.

XML Parser for C will check if an XML document is well-formed, and optionally validate it against a DTD. The parser constructs an object tree which can be accessed through a DOM interface or operate serially via a SAX interface.

You can post questions, comments, or bug reports to the XML Discussion Forum at <http://otn.oracle.com/tech/xml>.

Specifications

See [Appendix E, "XDK for C: Specifications and Cheat Sheets"](#) for a brief list of XML Parser for C methods and specifications.

See Also:

- The doc directory in your install area
- *Oracle9i XML Reference*
- On OTN under <http://otn.oracle.com/tech/xml/>

Memory Allocation

The memory callback functions `memcb` may be used if you wish to use your own memory allocation. If they are used, all of the functions should be specified.

The 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:

- `xmlparse()` or `xmlparsebuf()` is called to parse another file or buffer.
- `xmlclean()` is called.
- `xmlterm()` is called.

Thread Safety

If threads are forked off somewhere in the midst of the init-parse-term sequence of calls, you will get unpredictable behavior and results.

Data Types Index

[Table 24–1](#) lists the datatypes used in XML Parser for C.

Table 24–1 *Datatypes Used in XML Parser for C*

DataType	Description
oratext	String pointer
xmlctx	Master XML context
xmlmemcb	Memory callback structure (optional)
xmlsaxcb	SAX callback structure (SAX only)
ub4	32-bit (or larger) unsigned integer
uword	Native unsigned integer

Error Message Files

Error message files are provided in the `mesg/` subdirectory. The messages files also exist in the `$ORACLE_HOME/oracore/mesg` directory. You may set the environment variable `ORA_XML_MESG` to point to the absolute path of the `mesg/` subdirectory although this not required.

Validation Modes

Available validation modes are described in [Chapter 20, "Using XML Parser for Java"](#), "Oracle XML Parsers Support Four Validation Modes" on page 20-5.

XML Parser for C Usage

[Figure 24-1](#) describes XML Parser for C calling sequence as follows:

1. `XMLinit()` function initializes the parsing process.
2. The parsed item can be an XML document (file) or string buffer. If the input is an XML document or file, it is parsed using the `xmlparser()` function. If the input is a string buffer, it is parsed using the `xmlparserbuf()` function.
3. DOM or SAX API:

DOM: If you are using the DOM interface, include the following steps:

- The `xmlparse()` or `xmlparseBuffer()` function calls `.getDocumentElement()`. If no other DOM functions are being applied, you can invoke `xmlterm()`.
- This optionally calls other DOM functions if required. These are typically Node or print functions. It outputs the DOM document.
- If complete, the process invokes `xmlterm()`
- You can optionally first invoke `xmlclean()` to clean up any data structures created during the parse process. You would then call `xmlterm()`

SAX: If you are using the SAX interface, include the following steps:

- Process the results of the parser from `xmlparse()` or `xmlparseBuf()` using callback functions.
 - Register the callback functions.
4. Optionally, use `xmlclean()` to clean up the memory and structures used during a parse, and go to Step 5. or return to Step 2.
 5. Terminate the parsing process with `xmlterm()`

XML Parser for C usage is further explained in [Figure 24-1](#).

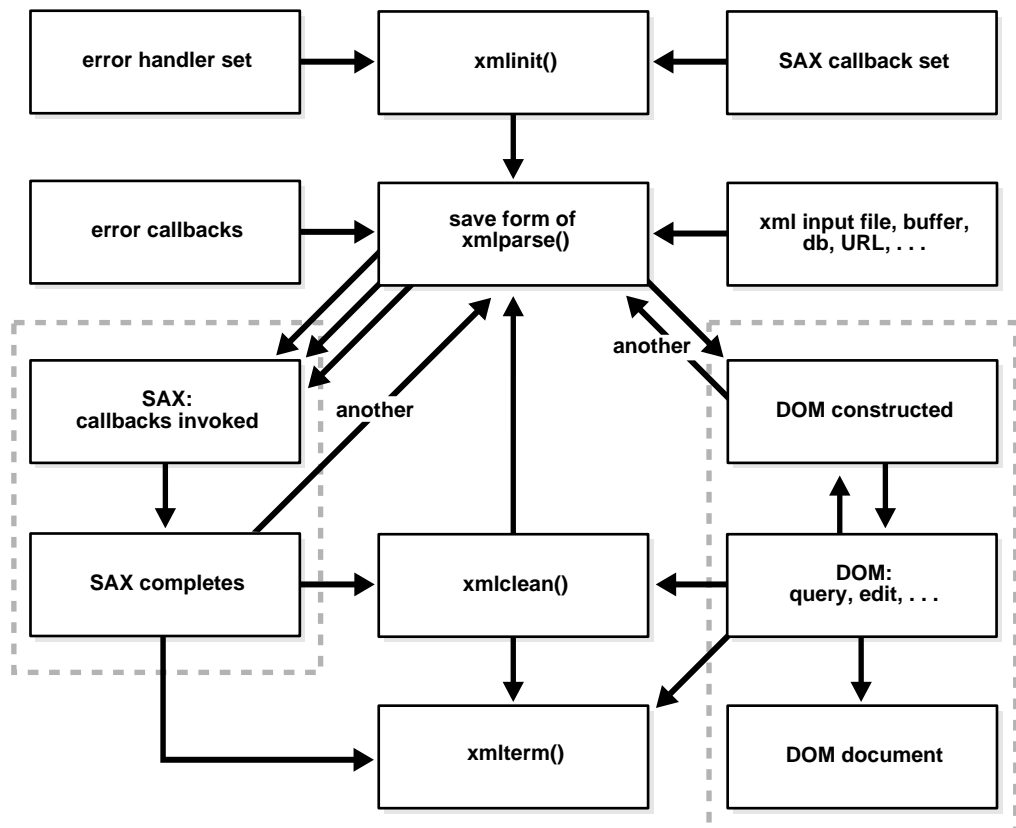
Parser Calling Sequence

The sequence of calls to the parser can be any of the following:

- `xmlinit()` - `xmlparse()` or `xmlparsebuf()` - `xmlterm()`
- `xmlinit()` - `xmlparse()` or

- ```
xmlparsebuf() - xmlclean() - xmlparse() or
xmlparsebuf() - xmlclean() -... - xmlterm()
```
- `xmlinit() - xmlparse() or`  
`xmlparsebuf() - xmlparse() or`  
`xmlparsebuf() -... - xmlterm()`

**Figure 24–1 XML Parser for C Calling Sequence**



## XML Parser for C, XSLT (DOM Interface) Usage

Figure 24-2 shows the XML Parser for C, XSLT functionality.

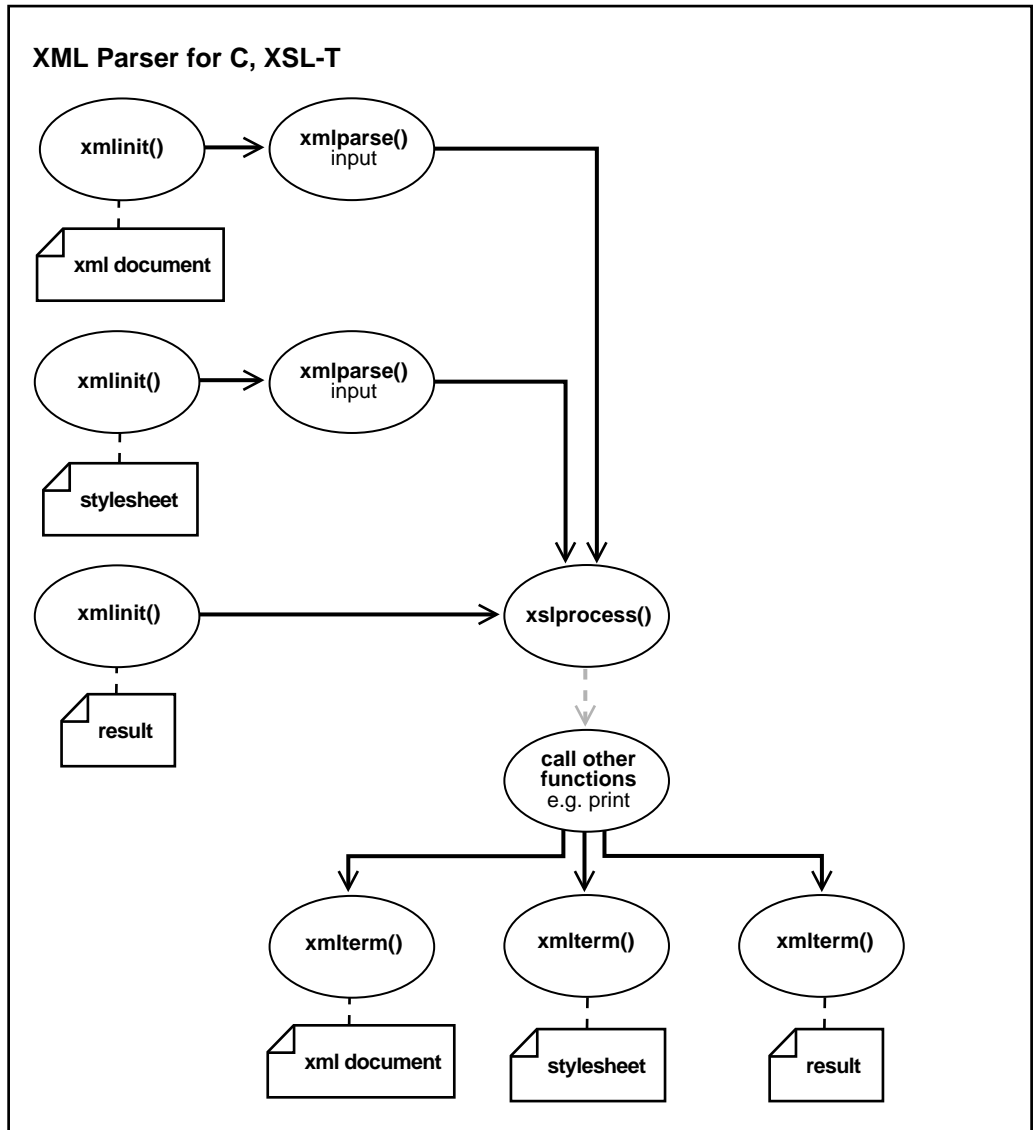
1. There are two inputs to `xmlparse()`:
  - The Stylesheet to be applied to the XML document
  - XML document

The output of `xmlparse()`, the parsed stylesheet and parsed XML document, are sent to the `xslprocess()` function for processing.
2. `xmlinit()` initializes the XSLT processing. `xmlinit()` initializes the `xslprocess()` result
3. `xslprocess()` optionally calls other functions, such as print functions. You can see the list of available functions either on OTN or in the *Oracle9i XML Reference*.
4. The resultant document (XML, HTML, VML, and so on) is typically sent to an application for further processing.
5. The application terminates the XSLT process by declaring `xmlterm()`, for the XML document, stylesheet, and final result.

XML Parser for C's XSLT functionality is illustrated with the following examples:

- [XML Parser for C Example 16: C — XSLSample.c](#) on page 24-52
- [XML Parser for C Example 17: C — XSLSample.std](#) on page 24-54

Figure 24-2 XML Parser for C: XSLT (DOM Interface) Usage



## XML Parser for C, Default Behavior

The following is the XML Parser for C default behavior:

- Character set encoding is UTF-8. If all your documents are ASCII, you are encouraged to set the encoding to US-ASCII for better performance.
- Messages are printed to stderr unless msghdlr is given.
- A parse tree which can be accessed by DOM APIs is built unless saxcb is set to use the SAX callback APIs. Note that any of the SAX callback functions can be set to NULL if not needed.
- The default behavior for the parser is to check that the input is well-formed but not to check whether it is valid. The flag XML\_FLAG\_VALIDATE can be set to validate the input. The default behavior for whitespace processing is to be fully conformant to the XML 1.0 spec, that is, all whitespace is reported back to the application but it is indicated which whitespace is ignorable. However, some applications may prefer to set the XML\_FLAG\_DISCARD\_WHITESPACE which will discard all whitespace between an end-element tag and the following start-element tag.

---

---

**Note:** It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

---

---

## DOM and SAX APIs

Oracle XML parser for C checks if an XML document is well-formed, and optionally validates it against a DTD. The parser constructs an object tree which can be accessed via one of the following interfaces:

- DOM interface
- Serially via a SAX interface

These two XML APIs:

- **DOM: Tree-based APIs.** A tree-based API compiles an XML document into an internal tree structure, then allows an application to navigate that tree using the Document Object Model (DOM), a standard tree-based API for XML and HTML documents.
- **SAX: Event-based APIs.** An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface.

Tree-based APIs are useful for a wide range of applications, but they often put a great strain on system resources, especially if the document is large (under very controlled circumstances, it is possible to construct the tree in a lazy fashion to avoid some of this problem). Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

In both of these cases, an event-based API provides a simpler, lower-level access to an XML document: you can parse documents much larger than your available system memory, and you can construct your own data structures using your callback event handlers.

## Using the SAX API

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 can also be included. That context pointer will be passed to each SAX function.

### SAX Callback Structure

The SAX callback structure:

```
typedef struct
```

```
{
 sword (*startDocument)(void *ctx);
 sword (*endDocument)(void *ctx);
 sword (*startElement)(void *ctx, const oratext *name,
 const struct xmlarray *attrs);
 sword (*endElement)(void *ctx, const oratext *name);
 sword (*characters)(void *ctx, const oratext *ch, size_t len);
 sword (*ignorableWhitespace)(void *ctx, const oratext *ch, size_t len);
 sword (*processingInstruction)(void *ctx, const oratext *target,
 const oratext *data);
 sword (*notationDecl)(void *ctx, const oratext *name,
 const oratext *publicId, const oratext *systemId);
 sword (*unparsedEntityDecl)(void *ctx, const oratext *name,
 const oratext *publicId,
 const oratext *systemId, const oratext *notationName);
 sword (*nsStartElement)(void *ctx, const oratext *qname,
 const oratext *local, const oratext *nsp,
 const struct xmlnodes *attrs);
} xmlsaxcb;
```

## Using the DOM API

See "[XML Parser for C Example 7: C — DOMSample.std](#)" on page 24-18.

## Invoking XML Parser for C

XML Parser for C can be invoked in two ways:

- By invoking the executable on the command line
- By writing C code and using the supplied APIs

### Command Line Usage

The XML Parser for C can be called as an executable by invoking `bin/xml`

[Table 24-2](#) lists the command line options.

**Table 24-2 XML Parser for C: Command Line Options**

| Option      | Description                                         |
|-------------|-----------------------------------------------------|
| -c          | Conformance check only, no validation               |
| -e encoding | Specify input file encoding                         |
| -h          | Help - show this usage help                         |
| -n          | Number - DOM traverse and report number of elements |
| -p          | Print document and DTD structures after parse       |
| -x          | Exercise SAX interface and print document           |
| -v          | Version - display parser version then exit          |
| -w          | Whitespace - preserve all whitespace                |

### Writing C Code to Use Supplied APIs

XML Parser for C can also be invoked by writing code to use the supplied APIs. The code must be compiled using the headers in the `include/` subdirectory and linked against the libraries in the `lib/` subdirectory. Please see the `Makefile` in the `sample/` subdirectory for full details of how to build your program.

## Using the Sample Files Included with Your Software

`$ORACLE_HOME/xdk/c/parser/sample/` directory contains several XML applications to illustrate how to use the XML Parser for C with the DOM and SAX interfaces.

[Table 24-3](#) lists the sample files in `sample/` directory.

**Table 24-3 XML Parser for C sample/ Files**

| <b>sample/ File Name</b> | <b>Description</b>                                                       |
|--------------------------|--------------------------------------------------------------------------|
| DOMNamespace.c           | Source for DOMNamespace program                                          |
| DOMNamespace.std         | Expected output from DOMNamespace                                        |
| DOMSample.c              | Source for DOMSample program                                             |
| DOMSample.std            | Expected output from DOMSample                                           |
| FullDOM.c                | Sample usage of DOM interface                                            |
| FullDOM.std              | Expected output from FullDOM                                             |
| Make.bat                 | Batch file for building sample programs                                  |
| NSExample.xml            | Sample XML file using namespaces                                         |
| SAXNamespace.c           | Source for SAXNamespace program                                          |
| SAXNamespace.std         | Expected output from SAXNamespace                                        |
| SAXSample.c              | Source for SAXSample program                                             |
| SAXSample.std            | Expected output from SAXSample                                           |
| XSLSample.c              | Source for XSLSample program                                             |
| XSLSample.std            | Expected output from XSLSample                                           |
| class.xml                | XML file that may be used with XSLSample                                 |
| iden.xsl                 | Stylesheet that may be used with XSLSample                               |
| cleo.xml                 | The Tragedy of Antony and Cleopatra<br>XML version of Shakespeare's play |



## Running the XML Parser for C Sample Programs

### Building the Sample programs

Change directories to `..sample/` and read the README file. This will explain how to build the sample programs according to your platform.

### Sample Programs

Table 24–4 lists the programs built by the sample files in `sample/`

**Table 24–4 XML Parser for C: Sample Built Programs in `sample/`**

| Built Program                | Description                                                                                                                                                |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DOMSample                    | A sample application using DOM APIs (shows an outline of Cleopatra, i.e. the XML elements ACT and SCENE).                                                  |
| SAXSample [word]             | A sample application using SAX APIs. Given a word, shows all lines in the play Cleopatra containing that word. If no word is specified, 'death' is used.   |
| DOMNamespace                 | Same as SAXNamespace except using DOM interface.                                                                                                           |
| SAXNamespace                 | A sample application using Namespace extensions to SAX API; prints out all elements and attributes of NSExample.xml along with full namespace information. |
| FullDOM                      | Sample usage of full DOM interface. Exercises all the calls, but does nothing too exciting.                                                                |
| XSLSample <xmlfile> <xsl ss> | Sample usage of XSL processor. It takes two filenames as input, the XML file and XSL stylesheet                                                            |

### XML Parser for C Example 1: XML — `class.xml`

`class.xml` is an XML file that inputs `XSLSample.c`.

```
<?xml version = "1.0"?>
<!DOCTYPE course [
<!ELEMENT course (Name, Dept, Instructor, Student)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Instructor (Name)>
<!ELEMENT Student (Name*)>
]>
<course>
```

```

<Name>Calculus</Name>
<Dept>Math</Dept>
<Instructor>
<Name>Jim Green</Name>
</Instructor>
<Student>
<Name>Jack</Name>
<Name>Mary</Name>
<Name>Paul</Name>
</Student>
</course>

```

## XML Parser for C Example 2: XML — cleo.xml

This XML example inputs DOMSample.c and SAXSample.c.

```

<?xml version="1.0"?>
<!DOCTYPE PLAY [
 <!ELEMENT PLAY (TITLE, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
PROLOGUE?, ACT+, EPILOGUE?)>
 <!ELEMENT TITLE (#PCDATA)>
 <!ELEMENT FM (P+)>
 <!ELEMENT P (#PCDATA)>
 <!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
 <!ELEMENT PGROUP (PERSONA+, GRPDESCR)>
 <!ELEMENT PERSONA (#PCDATA)>
 <!ELEMENT GRPDESCR (#PCDATA)>
 <!ELEMENT SCNDESCR (#PCDATA)>
 <!ELEMENT PLAYSUBT (#PCDATA)>
 <!ELEMENT INDUCT (TITLE, SUBTITLE*, (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
 <!ELEMENT ACT (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
 <!ELEMENT SCENE (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
 <!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
 <!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
 <!ELEMENT SPEECH (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
 <!ELEMENT SPEAKER (#PCDATA)>
 <!ELEMENT LINE (#PCDATA | STAGEDIR)*>
 <!ELEMENT STAGEDIR (#PCDATA)>
 <!ELEMENT SUBTITLE (#PCDATA)>
 <!ELEMENT SUBHEAD (#PCDATA)>
]>

<PLAY>
<TITLE>The Tragedy of Antony and Cleopatra</TITLE>

```

```
<PERSONAE>
<TITLE>Dramatis Personae</TITLE>

<PGROUP>
<PERSONA>MARK ANTONY</PERSONA>
<PERSONA>OCTAVIUS CAESAR</PERSONA>
<PERSONA>M. AEMILIUS LEPIDUS</PERSONA>
<GRPDESCR>triumvirs.</GRPDESCR>
</PGROUP>

<PERSONA>SEXTUS POMPEIUS</PERSONA>

<PGROUP>
<PERSONA>DOMITIUS ENOBARBUS</PERSONA>
<PERSONA>VENTIDIUS</PERSONA>
<PERSONA>EROS</PERSONA>
<PERSONA>SCARUS</PERSONA>
<PERSONA>DERCETAS</PERSONA>
<PERSONA>DEMETRIUS</PERSONA>
<PERSONA>PHILO</PERSONA>
<GRPDESCR>friends to Antony.</GRPDESCR>
</PGROUP>

<PGROUP>
<PERSONA>MECAENAS</PERSONA>
<PERSONA>AGRIPPA</PERSONA>
<PERSONA>DOLABELLA</PERSONA>
<PERSONA>PROCULEIUS</PERSONA>
<PERSONA>THYREUS</PERSONA>
<PERSONA>GALLUS</PERSONA>
<PERSONA>MENAS</PERSONA>
<GRPDESCR>friends to Caesar.</GRPDESCR>
</PGROUP>

...
...

<SCNDESCR>SCENE In several parts of the Roman empire.</SCNDESCR>

<PLAYSUBT>ANTONY AND CLEOPATRA</PLAYSUBT>

<ACT><TITLE>ACT I</TITLE>
```

```
<SCENE><TITLE>SCENE I. Alexandria. A room in CLEOPATRA's palace.</TITLE>
<STAGEDIR>Enter DEMETRIUS and PHILO</STAGEDIR>
```

```
<SPEECH>
<SPEAKER>PHILO</SPEAKER>
<LINE>Nay, but this dotage of our general's</LINE>
<LINE>O'erflows the measure: those his goodly eyes,</LINE>
<LINE>That o'er the files and musters of the war</LINE>
<LINE>Have glow'd like plated Mars, now bend, now turn,</LINE>
<LINE>The office and devotion of their view</LINE>
<LINE>Upon a tawny front: his captain's heart,</LINE>
<LINE>Which in the scuffles of great fights hath burst</LINE>
<LINE>The buckles on his breast, reneges all temper,</LINE>
<LINE>And is become the bellows and the fan</LINE>
<LINE>To cool a gipsy's lust.</LINE>
<STAGEDIR>Flourish. Enter ANTONY, CLEOPATRA, her Ladies,
the Train, with Eunuchs fanning her</STAGEDIR>
<LINE>Look, where they come:</LINE>
<LINE>Take but good note, and you shall see in him.</LINE>
<LINE>The triple pillar of the world transform'd</LINE>
<LINE>Into a strumpet's fool: behold and see.</LINE>
</SPEECH>
```

```
<SPEECH>
<SPEAKER>CLEOPATRA</SPEAKER>
<LINE>If it be love indeed, tell me how much.</LINE>
</SPEECH>
```

```
<SPEECH>
<SPEAKER>MARK ANTONY</SPEAKER>
<LINE>There's beggary in the love that can be reckon'd.</LINE>
</SPEECH>
```

```
<SPEECH>
<SPEAKER>CLEOPATRA</SPEAKER>
<LINE>I'll set a bourn how far to be beloved.</LINE>
</SPEECH>
```

```
<SPEECH>
<SPEAKER>MARK ANTONY</SPEAKER>
<LINE>Then must thou needs find out new heaven, new earth.</LINE>
</SPEECH>
```

```
...
...
```

```

...
<SPEAKER>DOLABELLA</SPEAKER>
<LINE>Here, on her breast,</LINE>
<LINE>There is a vent of blood and something blown:</LINE>
<LINE>The like is on her arm.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>First Guard</SPEAKER>
<LINE>This is an aspic's trail: and these fig-leaves</LINE>
<LINE>Have slime upon them, such as the aspic leaves</LINE>
<LINE>Upon the caves of Nile.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>OCTAVIUS CAESAR</SPEAKER>
<LINE>Most probable</LINE>
<LINE>That so she died; for her physician tells me</LINE>
<LINE>She hath pursued conclusions infinite</LINE>
<LINE>Of easy ways to die. Take up her bed;</LINE>
<LINE>And bear her women from the monument:</LINE>
<LINE>She shall be buried by her Antony:</LINE>
<LINE>No grave upon the earth shall clip in it</LINE>
<LINE>A pair so famous. High events as these</LINE>
<LINE>Strike those that make them; and their story is</LINE>
<LINE>No less in pity than his glory which</LINE>
<LINE>Brought them to be lamented. Our army shall</LINE>
<LINE>In solemn show attend this funeral;</LINE>
<LINE>And then to Rome. Come, Dolabella, see</LINE>
<LINE>High order in this great solemnity.</LINE>
</SPEECH>

<STAGEDIR>Exeunt</STAGEDIR>
</SCENE>
</ACT>
</PLAY>

```

## XML Parser for C Example 3: XSL — iden.xsl

This example stylesheet can be used to input `XSLSample.c`.

```

<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

```

```

 <xsl:template match="*|*|comment()|processing-instruction()|text()">
 <xsl:copy>
 <xsl:apply-templates
select="*|*|comment()|processing-instruction()|text()"/>
 </xsl:copy>
 </xsl:template>

</xsl:stylesheet>

```

## XML Parser for C Example 4: XML — FullDOM.xml (DTD)

This DTD example inputs FullDOM.c.

```

<!DOCTYPE doc [
 <!ELEMENT p (#PCDATA)>
 <!ATTLIST p xml:space (preserve|default) 'preserve'>
 <!NOTATION notation1 SYSTEM "file.txt">
 <!NOTATION notation2 PUBLIC "some notation">
 <!ELEMENT doc (p*)>
 <!ENTITY example "<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;);.</p>">
]>
<doc xml:lang="foo">&example;</doc>

```

## XML Parser for C Example 5: XML — NSEExample.xml

The following example file, NSEExample.xml, uses namespaces.

```

<!DOCTYPE doc [
<!ELEMENT doc (child*)>
<ATTLIST doc xmlns:nsprefix CDATA #IMPLIED>
<ATTLIST doc xmlns CDATA #IMPLIED>
<ATTLIST doc nsprefix:al CDATA #IMPLIED>
<!ELEMENT child (#PCDATA)>
]>
<doc nsprefix:al = "v1" xmlns="http://www.w3c.org"
xmlns:nsprefix="http://www.oracle.com">
<child>
This element inherits the default Namespace of doc.
</child>
</doc>

```

## XML Parser for C Example 6: C — DOMSample.c

This example contains the C source code for `DOMSample.c`

```

/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */
/*
 NAME
 DOMSample.c - Sample DOM usage
 DESCRIPTION
 Sample usage of C XML parser via DOM interface
*/

#include <stdio.h>

#ifdef ORATYPES
include <oratypes.h>
#endif
#ifdef ORAXML_ORACLE
include <oraxml.h>
#endif

#define DOCUMENT (oratext *) "cleo.xml"

void dump(xmlctx *ctx, xmlnode *node);
void dumppart(xmlctx *ctx, xmlnode *node, boolean indent);

int main()
{
 xmlctx *ctx;
 uword ecode;

 puts("XML C DOM sample");
 puts("Initializing XML package...");
 if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
 (void (*)(void *, const oratext *, uword)) 0,
 (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
 (const xmlmemcb *) 0, (void *) 0,
 (const oratext *) 0)))
 {
 printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
 return 1;
 }
}

```

```

 printf("Parsing '%s' ...\n", DOCUMENT);
 if (ecode = xmlparse(ctx, DOCUMENT, (oratext *) 0,
XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE))
 {
 printf("Parse failed, error %u\n", (unsigned) ecode);
 return 1;
 }

 puts("Outlining...");
 dump(ctx, getDocumentElement(ctx));

 xmlterm(ctx);

 return 0;
 }

void dump(xmlctx *ctx, xmlnode *node)
{
 const oratext *name;
 void *nodes;
 uword i, n_nodes;

 name = getNodeName(node);
 if (!strcmp((char *) name, "ACT"))
 dumppart(ctx, node, FALSE);
 else if (!strcmp((char *) name, "SCENE"))
 dumppart(ctx, node, TRUE);
 if (hasChildNodes(node))
 {
 nodes = getChildNodes(node);
 n_nodes = numChildNodes(nodes);
 for (i = 0; i < n_nodes; i++)
 dump(ctx, getChildNode(nodes, i));
 }
}

void dumppart(xmlctx *ctx, xmlnode *node, boolean indent)
{
 void *title = getFirstChild(node);

 if (indent)
 fputs(" ", stdout);
 puts((char *) getNodeValue(getFirstChild(title)));
}

```



```
/* end of DOMSample.c */
```

## XML Parser for C Example 7: C — DOMSample.std

The DOMSample.std example file shows the expected output from DOMSample.c

```
XML C DOM sample
Initializing XML package...
Parsing 'cleo.xml' ...
Outlining...
ACT I
 SCENE I. Alexandria. A room in CLEOPATRA's palace.
 SCENE II. The same. Another room.
 SCENE III. The same. Another room.
 SCENE IV. Rome. OCTAVIUS CAESAR's house.
 SCENE V. Alexandria. CLEOPATRA's palace.
ACT II
 SCENE I. Messina. POMPEY's house.
 SCENE II. Rome. The house of LEPIDUS.
...
...
...
ACT V
 SCENE I. Alexandria. OCTAVIUS CAESAR's camp.
 SCENE II. Alexandria. A room in the monument.
```

## XML Parser for C Example 8: C — SAXSample.c

This example contains the C source code for SAXSample.c

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/*
NAME
 SAXSample.c - Sample SAX usage

DESCRIPTION
 Sample usage of C XML parser via SAX interface
*/

#include <stdio.h>

#ifdef ORATYPES
```

```
include <oratypes.h>
#endif

#ifndef ORAXML_ORACLE
include <oraxml.h>
#endif

#define DOCUMENT"cleo.xml"
#define DEFAULT_KEYWORD"death"

char *keyword;
size_t keylen;
oraxml_t *elem;
oraxml_t speaker[80];

oraxml_t *findsub(oraxml_t *buf, size_t bufsiz, oraxml_t *sub, size_t subsiz);
void savestr(oraxml_t *buf, oraxml_t *s, size_t len);

/* SAX callback functions */

sword startDocument(void *ctx);
sword endDocument(void *ctx);
sword startElement(void *ctx, const oraxml_t *name,
 const struct xmlnodes *attrs);
sword endElement(void *ctx, const oraxml_t *name);
sword characters(void *ctx, const oraxml_t *ch, size_t len);

xmlsaxcb saxcb = {
 startDocument,
 endDocument,
 startElement,
 endElement,
 characters
};

int main(int argc, char **argv)
{
 xmlctx *ctx;
 ub4 flags;
 uword ecode;
 flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

 puts("XML C SAX sample");
 keyword = (argc > 1) ? argv[1] : DEFAULT_KEYWORD;
 keylen = strlen(keyword);
```

```
 puts("Initializing XML package...");

 if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
(void (*)(void *, const oratext *, uword)) 0,
(void *) 0, &saxcb, (void *) 0,
(const xmlmemb *) 0, (void *) 0,
 (const oratext *) 0)))
 {
 (void) printf("Failed to initialize XML parser, error %u\n",
 (unsigned) ecode);
 return 1;
 }

 printf("Parsing '%s' and looking for lines containing '%s'...\n",
DOCUMENT, keyword);
 elem = (oratext *) "";
 if (ecode = xmlparse(ctx, (oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;

 (void) xmlterm(ctx); /* terminate XML package */

 return 0;
}

sword startDocument(void *ctx)
{
 puts("startDocument");
 return 0;
}

sword endDocument(void *ctx)
{
 puts("endDocument");
 return 0;
}

sword startElement(void *ctx, const oratext *name,
 const struct xmlnodes *attrs)
{
 elem = (oratext *) name;
 return 0;
}

sword endElement(void *ctx, const oratext *name)
{

```

```

 elem = (orertext *) "";
 return 0;
 }

sword characters(void *ctx, const orertext *ch, size_t len)
{
 if (!strcmp((char *) elem, "SPEAKER"))
 savestr(speaker, (orertext *) ch, len);
 else if (findsub((orertext *) ch, len, (orertext *) keyword, keylen))
 printf(" %s: %.*s\n", speaker, len, ch);
 return 0;
}

orertext *findsub(orertext *buf, size_t bufsiz, orertext *sub, size_t subsiz)
{
 uword i;

 if (!buf || !bufsiz || (subsiz > bufsiz))
 return (orertext *) 0;
 if (!sub || !subsiz)
 return buf;
 for (i = 0; i < bufsiz - subsiz; i++, buf++)
 {
 if (!memcmp(buf, sub, subsiz))
 return buf;
 }
 return (orertext *) 0;
}

void savestr(orertext *buf, orertext *s, size_t len)
{
 memcpy(buf, s, len);
 buf[len] = 0;
}

/* End of SAXSample.c */

```

## XML Parser for C Example 9: C — SAXSample.std

SAXSample.std shows the expected output from SAXSample.c.

```

XML C SAX sample
Initializing XML package...
Parsing 'cleo.xml' and looking for lines containing 'death'...

```

```

startDocument
 MARK ANTONY: Who tells me true, though in his tale lie death,
 DOMITIUS ENOBARBUS: if they suffer our departure, death's the word.
 DOMITIUS ENOBARBUS: mettle in death, which commits some loving act upon
 MARK ANTONY: The death of Fulvia, with more urgent touches,
 MARK ANTONY: Is Fulvia's death.
 CLEOPATRA: In Fulvia's death, how mine received shall be.
 EROS: the poor third is up, till death enlarge his confine.
 SCARUS: Where death is sure. Yon ribaudred nag of Egypt,--
 EROS: Her head's declined, and death will seize her, but
 MARK ANTONY: I'll make death love me; for I will contend
 MARK ANTONY: Married to your good service, stay till death:
 MARK ANTONY: Than death and honour. Let's to supper, come,
 First Soldier: The hand of death hath raught him.
 CLEOPATRA: And bring me how he takes my death.
 MARK ANTONY: She hath betray'd me and shall die the death.
 MARK ANTONY: Than she which by her death our Caesar tells
 EROS: Of Antony's death.
 MARK ANTONY: A bridegroom in my death, and run into't
 DERCEAS: Thy death and fortunes bid thy followers fly.
 MARK ANTONY: Sufficing strokes for death.
 DIOMEDES: His death's upon him, but not dead.
 MARK ANTONY: I here importune death awhile, until
 CLEOPATRA: To rush into the secret house of death,
 CLEOPATRA: Ere death dare come to us? How do you, women?
 CLEOPATRA: And make death proud to take us. Come, away:
 OCTAVIUS CAESAR: And citizens to their dens: the death of Antony
 CLEOPATRA: What, of death too,
 CLEOPATRA: Where art thou, death?
 CLEOPATRA: The stroke of death is as a lover's pinch,
 CHARMIAN: Now boast thee, death, in thy possession lies
 OCTAVIUS CAESAR: Took her own way. The manner of their deaths?
endDocument

```

## XML Parser for C Example 10: C — DOMNamespace.c

This example contains the C source code for `DOMNamespace.c`.

```

/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/**
** This file demonstrates a simple use of the parser and Namespace
** extensions to the DOM APIs.
** The XML file that is given to the application is parsed and the

```

```

 ** elements and attributes in the document are printed.
 **/

#ifdef ORATYPES
include <oratypes.h>
#endif

#ifdef ORAXML_ORACLE
include <oraxml.h>
#endif

#define DOCUMENT "NSExample.xml"

/*-----
 FUNCTION PROTOTYPES
-----*/
static void DOMNSprint(xmlctx *ctx);
static void printElements(xmlctx *ctx, xmlnode *n);
static void printAttrs(xmlctx *ctx, xmlnode *n);

/*-----
 MAIN
-----*/
int main()
{
 xmlctx *ctx;
 oratext *encoding, *doc;
 void *saxcbctx;
 const xmlsaxcb *saxcb;
 uword ecode;
 ub4 flags;

 encoding = doc = (oratext *) 0;
 saxcbctx = (void *) 0;
 saxcb = (const xmlsaxcb *) 0;
 flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;
 doc = (oratext *)DOCUMENT;

 /* initialize LPX context */
 if (!(ctx = xmlinit(&ecode, encoding,
 (void (*)(void *, const oratext *, uword)) 0,
 (void *) 0, saxcb, saxcbctx, (const xmlmemcb *) 0,
 (void *) 0, (const oratext *) 0)))
 {
 printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
 }
}

```

```

 return -1;
 }

 /* parse the document */

 printf("\nParsing '%s' ...\n", doc);
 ecode = xmlparse(ctx, doc, encoding, flags);

 if (ecode)
 printf("Parse failed, code %u\n", (unsigned) ecode);
 else
 printf("Parse succeeded.\n");

 /* print results */

 printf("Printing results ...\n");
 DOMNSprint(ctx);

 /* terminate */

 (void) xmlterm(ctx);

 return (ecode ? -1 : 0);
}

/*-----
 DOMNSprint
-----*/
static void DOMNSprint(xmlctx *ctx)
{
 xmlnode *root;

 root = getDocumentElement(ctx);
 printf("\nThe elements are:\n");
 printElements(ctx, root);
}

/*-----
 printElements
-----*/
static void printElements(xmlctx *ctx, xmlnode *n)
{
 xmlnodes *nodes;
 uword i;
 size_t m;

```

```
const oratext *qname;
const oratext *namespace;
const oratext *local;
const oratext *prefix;

if (n == (xmlnode*)NULL)
 return;

if (nodes = getChildNodes(n))
{
 for (nn = numChildNodes(nodes), i = 0; i < nn; i++)
 {
 /* get node qualified name, local name, namespace, and prefix */

 qname = namespace = local = prefix = (oratext*)" ";

 if (getNodeQualifiedNodeName(n) != (oratext*)NULL)
 qname = getNodeQualifiedNodeName(n);
 if (getNodePrefix(n) != (oratext*)NULL)
 prefix = getNodePrefix(n);

 if (getNodeLocalName(n) != (oratext *)NULL)
 local = getNodeLocalName(n);

 if (getNodeNamespace(n) != (oratext*)NULL)
 namespace = getNodeNamespace(n);

 printf(" ELEMENT Qualified Name: %s\n", qname);
 printf(" ELEMENT Prefix Name : %s\n", prefix);
 printf(" ELEMENT Local Name : %s\n", local);
 printf(" ELEMENT Namespace : %s\n", namespace);

 printAttrs(ctx, n);
 printElements(ctx, (xmlnode *) getChildNode(nodes, i));
 }
}

/*-----
 printAttrs
-----*/
static void printAttrs(xmlctx *ctx, xmlnode *n)
{
 xmlnodes *attrs;
```



```
xmlnode *a;
uword i;
size_t na;

const oratext *value;
const oratext *qname;
const oratext *namespace;
const oratext *local;
const oratext *prefix;

if (attrs = getAttributes(n))
{
 printf("\n ATTRIBUTES: \n");
 for (na = numAttributes(attrs), i = 0; i < na; i++)
 {
 /* get attr qualified name, local name, namespace, and prefix */

 a = getAttributeIndex(attrs, i);

 qname = namespace = local = prefix = value = (oratext*)" ";

 if (getAttrQualifiedName(a) != (oratext*)NULL)
 qname = getAttrQualifiedName(a);
 if (getAttrNamespace(a) != (oratext*)NULL)
 namespace = getAttrNamespace(a);
 if (getAttrLocal(a) != (oratext*)NULL)
 local = getAttrLocal(a);
 if (getAttrPrefix(a) != (oratext*)NULL)
 prefix = getAttrPrefix(a);
 if (getAttrValue(a) != (oratext*)NULL)
 value = getAttrValue(a);

 printf(" %s = %s\n", qname, value);
 printf(" Namespace : %s\n", namespace);
 printf(" Local Name: %s\n", local);
 printf(" Prefix : %s\n\n", prefix);
 }
}
printf("\n");
}
```

## XML Parser for C Example 11: C — DOMNamespace.std

DOMNamespace.std shows the expected output from DOMNamespace.c.

```
Parsing 'NSExample.xml' ...
Parse succeeded.
Printing results ...
```

The elements are:

```
ELEMENT Qualified Name: doc
ELEMENT Prefix Name :
ELEMENT Local Name : doc
ELEMENT Namespace : http://www.w3c.org
```

ATTRIBUTES:

```
 nsprefix:a1 = v1
 Namespace : http://www.oracle.com
 Local Name : a1
 Prefix : nsprefix
```

```
 xmlns = http://www.w3c.org
 Namespace :
 Local Name: xmlns
 Prefix :
```

```
 xmlns:nsprefix = http://www.oracle.com
 Namespace :
 Local Name : nsprefix
 Prefix : xmlns
```

```
ELEMENT Qualified Name: child
ELEMENT Prefix Name :
ELEMENT Local Name : child
ELEMENT Namespace : http://www.w3c.org
```

## XML Parser for C Example 12: C — SAXNamespace.c

This example contains the C source code for the SAXNamespace.c.

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */
```

```
/**
```

```
** This file demonstrates a simple use of the Namespace extensions to
```

```

** the SAX APIs.
**/

#include <stdio.h>

#ifdef ORATYPES
include <oratypes.h>
#endif

#ifdef ORAXML_ORACLE
include <oraxml.h>
#endif

#define DOCUMENT "NSExample.xml"

/*-----
 FUNCTION PROTOTYPES
-----*/

static int sax_startdocument(void *ctx);
static int sax_enddocument(void *ctx);
static int sax_endelement(void *ctx, const oratext *name);
static int sax_nsstartelement(void *ctx, const oratext *qname,
 const oratext *local,
 const oratext *namespace,
 const struct xmlnodes *attrs);

/* SAX callback structure */

xmlsaxcb sax_callback = {
 sax_startdocument,
 sax_enddocument,
 0,
 sax_endelement,
 0,
 0,
 0,
 0,
 0,
 sax_nsstartelement,
 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* SAX callback context */

```

```
typedef struct {
 xmlctx *ctx;
 uword depth;
} cbctx;

/*-----
 MAIN
-----*/

int main()
{
 xmlctx *ctx;
 uword i;
 oratext *doc, *encoding;
 xmlsaxcb *saxcb;
 cbctx saxctx;
 void *saxcbctx;
 ub4 flags;
 uword ecode;

 doc = encoding = (oratext *)0;
 flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

 doc = (oratext *)DOCUMENT;

 /* set up SAX callbacks */

 saxcb = &sax_callback;
 saxcbctx = (void *) &saxctx;

 /* initialize LPX context */
 if (!(ctx = xmlinit(&ecode, encoding,
 (void (*)(void *, const oratext *, uword)) 0,
 (void *) 0, saxcb, saxcbctx, (const xmlmemcb *) 0,
 (void *) 0, (const oratext *) 0)))
 {
 printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
 return -1;
 }

 /* parse the document */

 printf("\nParsing '%s' ...\n", doc);
 ecode = xmlparse(ctx, doc, encoding, flags);
}
```

```
 if (ecode)
 printf("\nParse failed, code %u\n", (unsigned) ecode);
 else
 printf("\nParse succeeded.\n");

 /* terminate */

 (void) xmlterm(ctx);

 return (ecode ? -1 : 0);
}

/*-----
 SAX Interface
-----*/

static int sax_startdocument(void *ctx)
{
 printf("\nStartDocument\n\n");
 return 0;
}

static int sax_enddocument(void *ctx)
{
 printf("\nEndDocument\n");
 return 0;
}

static int sax_endelement(void *ctx, const oratext *name)
{
 printf("\nELEMENT Name : %s\n", name);
 return 0;
}

static int sax_nsstartelement(void *ctx, const oratext *qname,
 const oratext *local,
 const oratext *namespace,
 const struct xmlnodes *attrs)
{
 cbctx *saxctx = (cbctx *) ctx;
```

```
xmlnode *attr;
size_t i;

const oratext *aqname;
const oratext *aprefix;
const oratext *alocal;
const oratext *anamespace;
const oratext *avalue;

/*
 * Use the functions getXXXQualifiedName(), getXXXLocalName(), and
 * getXXXNamespace() to get Namespace information.
 */

if (qname == (oratext*)NULL)
 qname = (oratext*)" ";
if (local == (oratext*)NULL)
 local = (oratext*)" ";
if (namespace == (oratext*)NULL)
 namespace = (oratext*)" ";

printf("ELEMENT Qualified Name: %s\n", qname);
printf("ELEMENT Local Name : %s\n", local);
printf("ELEMENT Namespace : %s\n", namespace);

if (attrs)
{
 for (i = 0; i < numAttributes(attrs); i++)
 {
 attr = getAttributeIndex(attrs, i);

 aqname = aprefix = alocal = anamespace = avalue = (oratext*)" ";

 if (getAttrQualified_name(attr) != (oratext*)NULL)
 aqname = getAttrQualified_name(attr);

 if (getAttrPrefix(attr) != (oratext*)NULL)
 aprefix = getAttrPrefix(attr);

 if (getAttrLocal(attr) != (oratext*)NULL)
 alocal = getAttrLocal(attr);

 if (getAttrNamespace(attr) != (oratext*)NULL)
 anamespace = getAttrNamespace(attr);
 }
}
```

```

 if (getAttrValue(attr) != (oratext*)NULL)
 avalue = getAttrValue(attr);

 printf(" ATTRIBUTE Qualified Name : %s\n", aqname);
 printf(" ATTRIBUTE Prefix : %s\n", aprefix);
 printf(" ATTRIBUTE Local Name : %s\n", alocal);
 printf(" ATTRIBUTE Namespace : %s\n", anamespace);
 printf(" ATTRIBUTE Value : %s\n", avalue);
 printf("\n");
 }
}
return 0;
}

```

## XML Parser for C Example 13: C — SAXNamespace.std

SAXNamespace.std shows the expected output from SAXNamespace.c

Parsing 'NSExample.xml' ...

StartDocument

```

ELEMENT Qualified Name: doc
ELEMENT Local Name : doc
ELEMENT Namespace : http://www.w3c.org
ATTRIBUTE Qualified Name : nsprefix:a1
ATTRIBUTE Prefix : nsprefix
ATTRIBUTE Local Name : a1
ATTRIBUTE Namespace : http://www.oracle.com
ATTRIBUTE Value : v1

ATTRIBUTE Qualified Name : xmlns
ATTRIBUTE Prefix :
ATTRIBUTE Local Name : xmlns
ATTRIBUTE Namespace :
ATTRIBUTE Value : http://www.w3c.org

ATTRIBUTE Qualified Name : xmlns:nsprefix
ATTRIBUTE Prefix : xmlns
ATTRIBUTE Local Name : nsprefix
ATTRIBUTE Namespace :
ATTRIBUTE Value : http://www.oracle.com

```

```
ELEMENT Qualified Name: child
ELEMENT Local Name : child
ELEMENT Namespace : http://www.w3c.org

ELEMENT Name : child

ELEMENT Name : doc

EndDocument

Parse succeeded.
```

## XML Parser for C Example 14: C — FullDOM.c

This example contains the C source code for FullDOM.c

```
/* Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved. */

/*
 NAME
 FullDOM.c

 DESCRIPTION
 Sample code to test full DOM interface
*/

#include <stdio.h>

#ifdef ORATYPES
include <oratypes.h>
#endif

#ifdef ORAXML_ORACLE
include <oraxml.h>
#endif

#define TEST_DOCUMENT(oratext *) "FullDOM.xml"

void dump(xmlnode *node, uword level);
void dumpnode(xmlnode *node, uword level);

static char *ntypename[] = {
 "0",
 "ELEMENT",
```



```

"ATTRIBUTE",
"TEXT",
"CDATA",
"ENTREF",
"ENTITY",
"PI",
"COMMENT",
"DOCUMENT",
"DTD",
"DOCFRAG",
"NOTATION"
};

#define FAIL { puts("Failed!"); exit(1); }

int main()
{
 xmlctx *ctx;
 xmldtd *dtd;
 xmlnode *doc, *elem, *node, *text, *pi, *comment, *entref,
 *subelem, *subtext, *cdata, *attr1, *attr2, *clone,
 *deep_clone, *frag, *fragelem, *fragtext, *sub2,
 *fish, *food, *gleep1, *gleep2, *repl;
 xmlnodes *subs, *nodes, *attrs, *notes, *entities;
 uword i, ecode, level;

 puts("XML C Full DOM test");

 puts("Initializing XML parser...");

 if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
 (void (*)(void *, const oratext *, uword)) 0,
 (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
 (const xmlmemcb *) 0, (void *) 0,
 (const oratext *) 0)))
 {
 printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
 return 1;
 }

 puts("\nCreating new document...");
 if (!(doc = createDocument(ctx)))
 FAIL

 puts("Document from root node:");

```

```

dump(getDocument(ctx), 0);

puts("\nCreating 'ROOT' element...");
if (!(elem = createElement(ctx, (oratext *) "ROOT")))
FAIL

puts("Setting as 'ROOT' element...");
if (!appendChild(ctx, doc, elem)
FAIL

puts("Document from 'ROOT' element:");
dump(getDocumentElement(ctx), 0);

puts("Adding 7 children to 'ROOT' element...");
if (!(text = createTextNode(ctx, (oratext *) "Gibberish")) ||
 !appendChild(ctx, elem, text))
FAIL

if (!(comment = createComment(ctx, (oratext*) "Bit warm today, innit?")) ||
 !appendChild(ctx, elem, comment))
FAIL

if (!(pi = createProcessingInstruction(ctx, (oratext *) "target",
(oratext *) "PI-contents")) ||
 !appendChild(ctx, elem, pi))
FAIL

if (!(cdata = createCDATASection(ctx, (oratext *) "See DATA")) ||
 !appendChild(ctx, elem, cdata))
FAIL

if (!(entref = createEntityReference(ctx, (oratext *) "EntRef")) ||
 !appendChild(ctx, elem, entref))
FAIL

if (!(fish = createElement(ctx, (oratext *) "FISH")) ||
!appendChild(ctx, elem, fish))
FAIL

if (!(food = createElement(ctx, (oratext *) "FOOD")) ||
!appendChild(ctx, elem, food))
FAIL

puts("Document from 'ROOT' element with its 7 children:");
dump(getDocumentElement(ctx), 0);

```

```

 puts("\nTesting node insertion...");
 puts("Adding 'Pre-Gibberish' text node and 'Ask about the weather' comment
node ...");
 if (!(node = createTextNode(ctx, (oratext *) "Pre-Gibberish")) ||
 !insertBefore(ctx, elem, node, text))
FAIL

 if (!(node = createComment(ctx, (oratext *) "Ask about the weather:")) ||
 !insertBefore(ctx, elem, node, comment))
FAIL

 puts("Document from 'ROOT' element:");
 dump(getDocumentElement(ctx), 0);

 puts("\nTesting node removal by name ...");
 puts("Removing 'FISH' element");
 if (!(nodes = getChildNodes(elem)) ||
!removeNamedItem(nodes, (oratext *) "FISH"))
FAIL

 puts("Document from 'ROOT' element:");
 dump(getDocumentElement(ctx), 0);

 puts("\nTesting nextSibling links starting at first child...");
 for (node = getFirstChild(elem); node; node = getNextSibling(node))
dump(node, 1);

 puts("\nTesting previousSibling links starting at last child...");
 for (node = getLastChild(elem); node; node = getPreviousSibling(node))
dump(node, 1);

 puts("\nTesting setting node value...");
 puts("Original node:");
 dump(pi, 1);
 setNodeValue(pi, (oratext *) "New PI contents");
 puts("Node after new value:");
 dump(pi, 1);

 puts("\nAdding another element level, i.e., 'SUB' ...");
 if (!(subelem = createElement(ctx, (oratext *) "SUB")) ||
!insertBefore(ctx, elem, subelem, cdata) ||
!(subtext = createTextNode(ctx, (oratext *) "Lengthy SubText")) ||
!appendChild(ctx, subelem, subtext))
FAIL

```

```
puts("Document from 'ROOT' element:");
dump(getDocumentElement(ctx), 0);

puts("\nAdding a second 'SUB' element...");
if (!(sub2 = createElement(ctx, (oratext *) "SUB")) ||
!insertBefore(ctx, elem, sub2, cdata))
FAIL

puts("Document from 'ROOT' element:");
dump(getDocumentElement(ctx), 0);

puts("\nGetting all SUB nodes - note the distinct hex addresses ...");
if (!(subs = getElementsByTagName(ctx, (xmlnode *) 0, (oratext *) "SUB")))
FAIL
for (i = 0; i < getNodeMapLength(subs); i++)
dumpnode(getChildNode(subs, i), 1);

puts("\nTesting parent links...");
for (level = 1, node = subtext; node; node = getParentNode(node), level++)
dumpnode(node, level);

puts("\nTesting owner document of node...");
dumpnode(subtext, 1);
dumpnode(getOwnerDocument(subtext), 1);

puts("\nTesting node replacement...");
if (!(node = createTextNode(ctx, (oratext *) "REPLACEMENT, 1/2 PRICE")) ||
!replaceChild(ctx, pi, node))
FAIL

puts("Document from 'ROOT' element:");
dump(getDocumentElement(ctx), 0);

puts("\nTesting node removal...");
if (!removeChild(entref))
FAIL

puts("Document from 'ROOT' element:");
dump(getDocumentElement(ctx), 0);

puts("\nNormalizing...");
normalize(ctx, elem);

puts("Document from 'ROOT' element:");
```

```

 dump(getDocumentElement(ctx), 0);

 puts("\nCreating and populating document fragment...");
 if (!(frag = createDocumentFragment(ctx)) ||
 !(fragelem = createElement(ctx, (oratext *) "FragElem")) ||
 !(fragtext = createTextNode(ctx, (oratext *) "FragText")) ||
 !appendChild(ctx, frag, fragelem) ||
 !appendChild(ctx, frag, fragtext))
 FAIL
 dump(frag, 1);

 puts("Insert document fragment...");
 if (!insertBefore(ctx, elem, frag, comment))
 FAIL
 dump(elem, 1);

 puts("\nCreate two attributes...");
 if (!(attr1 = createAttribute(ctx, (oratext*)"Attr1", (oratext*)"Value1")) ||
 !(attr2 = createAttribute(ctx, (oratext*)"Attr2", (oratext*)"Value2")))
 FAIL
 puts("Setting attributes...");
 if (!setAttributeNode(ctx, subelem, attr1, NULL) ||
 !setAttributeNode(ctx, subelem, attr2, NULL))
 FAIL
 dump(subelem, 1);

 puts("\nAltering attributel value...");
 setAttrValue(attr1, (oratext *) "New1");
 dump(subelem, 1);

 puts("\nFetching attribute by name (Attr2)...");
 if (!(node = getAttributeNode(subelem, (oratext *) "Attr2")))
 FAIL
 dump(node, 1);

 puts("\nRemoving attribute by name (Attr1)...");
 removeAttribute(subelem, (oratext *) "Attr1");
 dump(subelem, 1);

 puts("\nAdding new attribute...");
 if (!setAttribute(ctx, subelem, (oratext *) "Attr3", (oratext *) "Value3"))
 FAIL
 dump(subelem, 1);

 puts("\nRemoving attribute by pointer (Attr2)...");

```

```

 if (!removeAttributeNode(subelem, attr2))
FAIL
 dump(subelem, 1);

 puts("\nAdding new attribute w/same name (test replacement)...");
 dump(subelem, 1);
 if (!(node = createAttribute(ctx, (oratext*)"Attr3", (oratext*)"Zoo3")))
FAIL
 if (!setAttributeNode(ctx, subelem, node, NULL))
FAIL
 dump(subelem, 1);

 puts("\nTesting node (attr) set by name ...");
 puts("Adding 'GLEEP' attribute and printing out hex addresses of node set");
 attrs = getAttributes(subelem);
 if (!(gleep1=createAttribute(ctx,(oratext*)"GLEEP",(oratext*)"gleep1")) ||
!setNamedItem(ctx, attrs, gleep1, NULL))
FAIL
 dump(subelem, 1);

 puts("\nTesting node set by name ...");
 puts("Replacing 'GLEEP' attribute - note the changed hex address");
 if (!(gleep2=createAttribute(ctx,(oratext*)"GLEEP",(oratext*)"gleep2")) ||
!setNamedItem(ctx, attrs, gleep2, &repl))
FAIL
 dump(subelem, 1);
 puts("Replaced node was:");
 dump(repl, 1);

 puts("\nOriginal SubROOT...");
 dump(subelem, 1);
 puts("Cloned SubROOT (not deep)...");
 clone = cloneNode(ctx, subelem, FALSE);
 dump(clone, 1);
 puts("Cloned SubROOT (deep)...");
 deep_clone = cloneNode(ctx, subelem, TRUE);
 dump(deep_clone, 1);

 puts("\nSplitting text...");
 dump(subelem, 1);
 splitText(ctx, subtext, 3);
 dump(subelem, 1);

 puts("\nTesting string operations...");
 printf(" CharData = \"%s\"\n", getCharData(subtext));

```

```

 puts("Setting new data...");
 setCharData(subtext, (oratext *) "0123456789");
 printf(" CharData = \"%s\"\n", getCharData(subtext));
 printf(" CharLength = %d\n", getCharLength(subtext));
 printf(" Substring(0,5) = \"%s\"\n",
substringData(ctx, subtext, 0, 5));
 printf(" Substring(8,2) = \"%s\"\n",
substringData(ctx, subtext, 8, 2));
 puts("Appending data...");
 appendData(ctx, subtext, (oratext *) "ABCDEF");
 printf(" CharData = \"%s\"\n", getCharData(subtext));
 puts("Inserting data...");
 insertData(ctx, subtext, 10, (oratext *) "**foo*");
 printf(" CharData = \"%s\"\n", getCharData(subtext));
 puts("Deleting data...");
 deleteData(ctx, subtext, 0, 10);
 printf(" CharData = \"%s\"\n", getCharData(subtext));
 puts("Replacing data...");
 replaceData(ctx, subtext, 1, 3, (oratext *) "bamboozle");
 printf(" CharData = \"%s\"\n", getCharData(subtext));

 puts("Cleaning up...");
 xmlclean(ctx);

 if (getDocument(ctx))
 {
puts("Problem, document is not gone!!");
return 1;
 }

 puts("Parsing test document...");
 if (ecode = xmlparse(ctx, TEST_DOCUMENT, (oratext *) 0, 0))
 {
printf("Parse failed, code %d\n", (int) ecode);
return 1;
 }

 puts("Document from root node:");
 dump(getDocument(ctx), 0);

 dtd = getDocType(ctx);

 puts("Testing getDocTypeNotations...");
 if (notes = getDocTypeNotations(dtd))
 {

```

```
size_t n_notes = numChildNodes(notes);

printf("# of notations = %d\n", (int) n_notes);
for (i = 0; i < n_notes; i++)
 dump(getChildNode(notes, i), 1);
 }
 else
puts("No defined notations\n");

 puts("Testing getDocTypeEntities...");
 if (entities = getDocTypeEntities(dtd))
 {
size_t n_entities = numChildNodes(entities);

printf("# of entities = %d\n", (int) n_entities);
for (i = 0; i < n_entities; i++)
 dump(getChildNode(entities, i), 1);
 }
 else
puts("No defined entities\n");

 puts("Cleaning up...");
 xmlclean(ctx);

 if (getDocument(ctx))
 {
puts("Problem, document is not gone!!\n");
return 1;
 }

 puts("\nTerminating parser...");
 xmlterm(ctx);

 puts("Success.");
 return 0;
}

void dump(xmlnode *node, uword level)
{
 xmlnodes *nodes;
 uword i, n_nodes;

 if (node)
 {
dumpnode(node, level);
```



```

if (hasChildNodes(node))
{
 nodes = getChildNodes(node);
 n_nodes = numChildNodes(nodes);
 for (i = 0; i < n_nodes; i++)
dump(getChildNode(nodes, i), level + 1);
}
}

void dumpnode(xmlnode *node, uword level)
{
 const oratext *name, *value;
 xmlntype type;
 xmlnodes *attrs;
 xmlnode *attr;
 uword i, n_attrs;

 if (node)
 {
 for (i = 0; i <= level; i++)
 fputs(" ", stdout);
 type = getNodeType(node);
 fputs(ntypename[type], stdout);
 if ((name = getNodeName(node)) && (*name != '#'))
 printf(" \"%s\"", (char *) name);
 if (value = getNodeValue(node))
 printf(" = \"%s\"", (char *) value);
 if ((type == ELEMENT_NODE) && (attrs = getAttributes(node)))
 {
 fputs(" [", stdout);
 n_attrs = numAttributes(attrs);
 for (i = 0; i < n_attrs; i++)
 {
 if (i) fputs(", ", stdout);
 attr = getAttributeIndex(attrs, i);
 fputs((char *) getAttrName(attr), stdout);
 if (getAttrSpecified(attr))
 putchar('*');
 printf("=\"%s\"", (char *) getAttrValue(attr));
 }
 putchar(']');
 }
 putchar('\n');
 }
}

```

```
}

/* end of FullDOM.c */
```

## XML Parser for C Example 15: C — FullDOM.std

FullDOM.std shows the expected output from FullDOM.c.

```
XML C Full DOM test
Initializing XML parser...

Creating new document...
Document from root node:
 DOCUMENT

Creating 'ROOT' element...
Setting as 'ROOT' element...
Document from 'ROOT' element:
 ELEMENT "ROOT"
Adding 7 children to 'ROOT' element...
Document from 'ROOT' element with its 7 children:
 ELEMENT "ROOT"
 TEXT = "Gibberish"
 COMMENT = "Bit warm today, innit?"
 PI "target" = "PI-contents"
 CDATA = "See DATA"
 ENTREF "EntRef"
 ELEMENT "FISH"
 ELEMENT "FOOD"

Testing node insertion...
Adding 'Pre-Gibberish' text node and 'Ask about the weather' comment node ...
Document from 'ROOT' element:
 ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 PI "target" = "PI-contents"
 CDATA = "See DATA"
 ENTREF "EntRef"
 ELEMENT "FISH"
 ELEMENT "FOOD"
```

Testing node removal by name ...

Removing 'FISH' element

Document from 'ROOT' element:

```
ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 PI "target" = "PI-contents"
 CDATA = "See DATA"
 ENTREF "EntRef"
 ELEMENT "FOOD"
```

Testing nextSibling links starting at first child...

```
TEXT = "Pre-Gibberish"
TEXT = "Gibberish"
COMMENT = "Ask about the weather:"
COMMENT = "Bit warm today, innit?"
PI "target" = "PI-contents"
CDATA = "See DATA"
ENTREF "EntRef"
ELEMENT "FOOD"
```

Testing previousSibling links starting at last child...

```
ELEMENT "FOOD"
ENTREF "EntRef"
CDATA = "See DATA"
PI "target" = "PI-contents"
COMMENT = "Bit warm today, innit?"
COMMENT = "Ask about the weather:"
TEXT = "Gibberish"
TEXT = "Pre-Gibberish"
```

Testing setting node value...

Original node:

```
PI "target" = "PI-contents"
```

Node after new value:

```
PI "target" = "New PI contents"
```

Adding another element level, i.e., 'SUB' ...

Document from 'ROOT' element:

```
ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
```

```
COMMENT = "Bit warm today, innit?"
PI "target" = "New PI contents"
ELEMENT "SUB"
 TEXT = "Lengthy SubText"
CDATA = "See DATA"
ENTREF "EntRef"
ELEMENT "FOOD"
```

Adding a second 'SUB' element...

Document from 'ROOT' element:

```
ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 PI "target" = "New PI contents"
 ELEMENT "SUB"
 TEXT = "Lengthy SubText"
 ELEMENT "SUB"
 CDATA = "See DATA"
 ENTREF "EntRef"
 ELEMENT "FOOD"
```

Getting all SUB nodes - note the distinct hex addresses ...

```
ELEMENT "SUB"
ELEMENT "SUB"
```

Testing parent links...

```
TEXT = "Lengthy SubText"
 ELEMENT "SUB"
 ELEMENT "ROOT"
 DOCUMENT
```

Testing owner document of node...

```
TEXT = "Lengthy SubText"
DOCUMENT
```

Testing node replacement...

Document from 'ROOT' element:

```
ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 TEXT = "REPLACEMENT, 1/2 PRICE"
```

```

ELEMENT "SUB"
 TEXT = "Lengthy SubText"
ELEMENT "SUB"
CDATA = "See DATA"
ENTREF "EntRef"
ELEMENT "FOOD"

```

Testing node removal...

Document from 'ROOT' element:

```

ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 TEXT = "REPLACEMENT, 1/2 PRICE"
 ELEMENT "SUB"
 TEXT = "Lengthy SubText"
 ELEMENT "SUB"
 CDATA = "See DATA"
 ELEMENT "FOOD"

```

Normalizing...

Document from 'ROOT' element:

```

ELEMENT "ROOT"
 TEXT = "Pre-GibberishGibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 TEXT = "REPLACEMENT, 1/2 PRICE"
 ELEMENT "SUB"
 TEXT = "Lengthy SubText"
 ELEMENT "SUB"
 CDATA = "See DATA"
 ELEMENT "FOOD"

```

Creating and populating document fragment...

```

DOCFRAG
 ELEMENT "FragElem"
 TEXT = "FragText"

```

Insert document fragment...

```

ELEMENT "ROOT"
 TEXT = "Pre-GibberishGibberish"
 COMMENT = "Ask about the weather:"
 ELEMENT "FragElem"
 TEXT = "FragText"
 COMMENT = "Bit warm today, innit?"

```

```
TEXT = "REPLACEMENT, 1/2 PRICE"
ELEMENT "SUB"
 TEXT = "Lengthy SubText"
ELEMENT "SUB"
CDATA = "See DATA"
ELEMENT "FOOD"
```

Create two attributes...

Setting attributes...

```
ELEMENT "SUB" [Attr1*="Value1", Attr2*="Value2"]
 TEXT = "Lengthy SubText"
```

Altering attribute1 value...

```
ELEMENT "SUB" [Attr1*="New1", Attr2*="Value2"]
 TEXT = "Lengthy SubText"
```

Fetching attribute by name (Attr2)...

```
ATTRIBUTE "Attr2" = "Value2"
```

Removing attribute by name (Attr1)...

```
ELEMENT "SUB" [Attr2*="Value2"]
 TEXT = "Lengthy SubText"
```

Adding new attribute...

```
ELEMENT "SUB" [Attr2*="Value2", Attr3*="Value3"]
 TEXT = "Lengthy SubText"
```

Removing attribute by pointer (Attr2)...

```
ELEMENT "SUB" [Attr3*="Value3"]
 TEXT = "Lengthy SubText"
```

Adding new attribute w/same name (test replacement)...

```
ELEMENT "SUB" [Attr3*="Value3"]
 TEXT = "Lengthy SubText"
ELEMENT "SUB" [Attr3*="Zoo3"]
 TEXT = "Lengthy SubText"
```

Testing node (attr) set by name ...

Adding 'GLEEP' attribute and printing out hex addresses of node set

```
ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep1"]
 TEXT = "Lengthy SubText"
```

Testing node set by name ...

Replacing 'GLEEP' attribute - note the changed hex address

```
ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
```

```

 TEXT = "Lengthy SubText"
Replaced node was:
 ATTRIBUTE "GLEEP" = "gleep1"

Original SubROOT...
 ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
 TEXT = "Lengthy SubText"
Cloned SubROOT (not deep)...
 ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
 TEXT = "Lengthy SubText"
Cloned SubROOT (deep)...
 ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
 TEXT = "Lengthy SubText"

Splitting text...
 ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
 TEXT = "Lengthy SubText"
 ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="gleep2"]
 TEXT = "Leng"
 TEXT = "thy SubText"

Testing string operations...
 CharData = "Leng"
Setting new data...
 CharData = "0123456789"
 CharLength = 10
 Substring(0,5) = "01234"
 Substring(8,2) = "89"
Appending data...
 CharData = "0123456789ABCDEF"
Inserting data...
 CharData = "0123456789*foo*ABCDEF"
Deleting data...
 CharData = "*foo*ABCDEF"
Replacing data...
 CharData = "*bamboozle*ABCDEF"
Cleaning up...
Parsing test document...
Document from root node:
 DOCUMENT
 DTD "doc"
 ELEMENT "doc" [xml:lang*="foo"]
 ELEMENT "p" [xml:space="preserve"]
 TEXT = "An ampersand (&) may be escaped
numerically (&) or with a general entity

```

```
(&);."
Testing getDocTypeNotations...
of notations = 2
 NOTATION "notation1"
 NOTATION "notation2"
Testing getDocTypeEntities...
of entities = 1
 ENTITY "example" = "<p>An ampersand (&#38;) may be escaped
numerically (&#38;) or with a general entity
(&amp;);.</p>"
Cleaning up...

Terminating parser...
Success.
```

## XML Parser for C Example 16: C — XSLSample.c

This example contains C source code for XSLSample.c.

```
/* Copyright (c) Oracle Corporation 1999. All Rights Reserved. */

/*
 NAME
 XSLSample.c - Sample function for XSL
 DESCRIPTION
 Sample usage of C XSL Processor
*/

#include <stdio.h>
#ifdef ORATYPES
include <oratypes.h>
#endif

#ifdef ORAXML_ORACLE
include <oraxml.h>
#endif

int main(int argc, char *argv[])
{
 xmlctx *xctx, *xslctx, *resctx;
 xmlnode *result;
 uword ecode;
 /* Check for correct usage */
 if (argc < 3)
```



```

 {
 puts("Usage is XSLSample <xmlfile> <xslfile>\n");
 return 1;
 }

/* Parse the XML document */
if (!(xctx = xmlinit(&ecode, (const oratext *) 0,
 (void (*)(void *, const oratext *, uword)) 0,
 (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
 (const xmlmemcb *) 0, (void *) 0,
 (const oratext *) 0)))
{
 printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
 return 1;
}

printf("Parsing '%s' ...\n", argv[1]);
if (ecode = xmlparse(xctx, (oratext *)argv[1], (oratext *) 0,
 XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE))
{
 printf("Parse failed, error %u\n", (unsigned) ecode);
 return 1;
}

/* Parse the XSL document */
if (!(xslctx = xmlinit(&ecode, (const oratext *) 0,
 (void (*)(void *, const oratext *, uword)) 0,
 (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
 (const xmlmemcb *) 0, (void *) 0,
 (const oratext *) 0)))
{
 printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
 return 1;
}

printf("Parsing '%s' ...\n", argv[2]);
if (ecode = xmlparse(xslctx, (oratext *)argv[2], (oratext *) 0,
 XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE))
{
 printf("Parse failed, error %u\n", (unsigned) ecode);
 return 1;
}

/* Initialize the result context */
if (!(resctx = xmlinit(&ecode, (const oratext *) 0,

```

```
 (void (*)(void *, const oratext *, uword)) 0,
 (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
 (const xmlmemcb *) 0, (void *) 0,
 (const oratext *) 0)))
 {
 printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
 return 1;
 }

 /* XSL processing */
 printf("XSL Processing\n");
 if (ecode = xslprocess(xctx, xslctx, resctx, &result))
 {
 printf("Parse failed, error %u\n", (unsigned) ecode);
 return 1;
 }

 /* Print the result tree */
 printres(resctx, result);

 /* Call the terminate functions */
 (void)xmlterm(xctx);
 (void)xmlterm(xslctx);
 (void)xmlterm(resctx);

 return 0;
}
```

## XML Parser for C Example 17: C — XSLSample.std

XSLSample.std shows the expected output from XSLSample.c.

```
Parsing 'class.xml' ...
Parsing 'iden.xsl' ...
XSL Processing
<root>
 <course>
 <Name>Calculus</Name>
 <Dept>Math</Dept>
 <Instructor>
 <Name>Jim Green</Name>
 </Instructor>
 <Student>
 <Name>Jack</Name>
```

```
 <Name>Mary</Name>
 <Name>Paul</Name>
 </Student>
</course>
</root>
```



---

## Using XML Schema Processor for C

This chapter contains the following sections:

- [Oracle XML Schema Processor for C](#)
- [Invoking XML Schema Processor for C](#)
- [XML Schema Processor for C Usage Diagram](#)
- [How to Run XML Schema for C Sample Programs](#)

## Oracle XML Schema Processor for C

The XML Schema Processor for C is a companion component to the XML Parser for C. It allows support for simple and complex datatypes in Oracle XML applications.

XML Schema Processor for C supports the W3C XML Schema Working Draft, with the goal being that it be 100% fully conformant when XML Schema becomes a W3C Recommendation. This makes writing custom applications that process XML documents straightforward in the Oracle environment, and means that a standards-compliant XML Schema Processor is part of the Oracle platform on every operating system where Oracle is ported.

**See Also:** [Chapter 21, "Using XML Schema Processor for Java"](#), for more information about XML Schema and why you would want to use XML Schema.

## Oracle XML Schema for C Features

XML Schema Processor for C has the following features:

- Supports simple and complex types
- Built on XML Parser for C v2
- Supports the W3C XML Schema Working Drafts

**See Also:**

- *Oracle9i XML Reference*
- [Appendix E, "XDK for C: Specifications and Cheat Sheets"](#)

## Requirements

XML Schema Processor for C runs on the following operating systems:

- Linux
- Solaris
- HP-UX
- NT 4 / Service Pack 3 (and above)

### Online Documentation

Documentation for Oracle XML Schema Processor for C is located in the doc/ directory in your install area.

## Standards Conformance

Oracle XML Parser for C conforms to the following standards:

- W3C recommendation for Extensible Markup Language (XML) 1.0
- W3C recommendation for Document Object Model Level 1.0
- W3C proposed recommendation for Namespaces in XML
- Simple API for XML (SAX) 1.0
- W3C recommendation for XSL Transformations (XSLT)
- W3C recommendation for XML Path Language (XPath)

## Using the Supported Character Sets

The XML Parser for C currently supports the following encodings:

- BIG5
- EBCDIC-CP-BE
- EBCDIC-CP-CA
- EBCDIC-CP-CH
- EBCDIC-CP-DK
- EBCDIC-CP-ES
- EBCDIC-CP-FI
- EBCDIC-CP-FR
- EBCDIC-CP-GB
- EBCDIC-CP-HE
- EBCDIC-CP-IS
- EBCDIC-CP-IT
- EBCDIC-CP-NL
- EBCDIC-CP-NO
- EBCDIC-CP-ROECE
- EBCDIC-CP-SE
- EBCDIC-CP-US

- EBCDIC-CP-WT
- EBCDIC-CP-YU
- EUC-JP
- GB2312
- ISO-10646-UCS-2
- ISO-8859-1 through 9
- KOI8-RUTF-8
- SHIFT\_JIS
- US-ASCII
- UTF-16

**See Also:** Appendix A, Character Sets, of the *Oracle9i Globalization and National Language Support Guide*, where, in addition, any character set specified in can be used.

To use these encodings, you must have the following set:

- The ORACLE\_HOME environment variable must be set to point to the location of your Oracle installation.
- The environment variables, ORA\_NLS, ORA\_NLS32, and ORA\_NLS33, must be set to point to the location of the NLS data files.
  - On Unix systems, this is usually `$ORACLE_HOME/ocommon/nls/admin/data`.
  - On Windows NT, this is usually `$ORACLE_HOME\nlsrtl\admin\nlsdata`.

The default encoding is UTF-8. It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to twice as fast as with multibyte character sets, such as UTF-8.

## XML Schema Processor for C: Software

[Table 25-1](#) lists the supplied files and directories with this release:



**Table 25–1 XML Schema Processor for C: Supplied Files**

Directory and Files	Description
license.html	Licensing agreement
readme.html	This file
bin/	Schema processor executable, “schema”
doc/	API documentation
include/	header files
lib/	XML/XSL/Schema & support libraries
mesg/	Error message files
sample/	Example usage of the Schema processor

Table 25–2 lists the included libraries:

**Table 25–2 XML Schema Processor for C: Supplied Libraries**

Included Library	Description
libxml8.a	XML Parser/XSL Processor
libxsd8.a	XML Schema Processor
libcore8.a	CORE functions
libnls8.a	National Language Support

## Invoking XML Schema Processor for C

XML Schema Processor for C can be called as an executable by invoking `bin/schema` in the install area. This takes two arguments:

- XML instance document
- Optionally, a default schema

The Schema Processor can also be invoked by writing code using the supplied APIs. The code must be compiled using the headers in the `include/` subdirectory and linked against the libraries in the `lib/` subdirectory. See `Makefile` in the `sample/` subdirectory for details on how to build your program.

An error message file is provided in the `mesg/` subdirectory. Currently, the only message file is in English although message files for other languages may be supplied in future releases.

### Set Environment Variable `OR_XML_MESG` to Point to Absolute Path

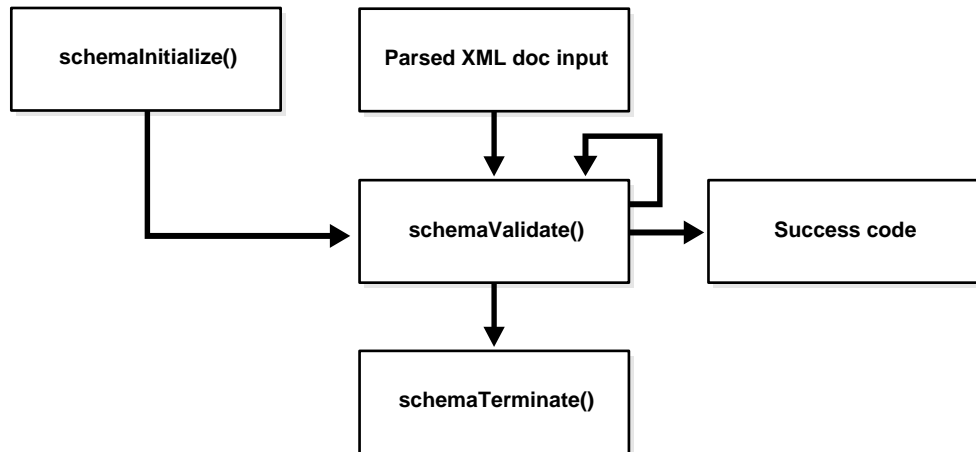
You should set the environment variable `ORA_XML_MESG` to point to the “**absolute**” path of the `mesg/` subdirectory. Alternately, if you have an `$ORACLE_HOME` installed, you may copy the contents of the `mesg/` subdirectory to the `$ORACLE_HOME/oracore/mesg` directory.

## XML Schema Processor for C Usage Diagram

[Figure 25-1](#) describes the calling sequence for the XML Schema Processor for C, as follows:

The sequence of calls to the processor is: initialize, validate, validate,...., validate, terminate.

1. The initialize call is invoked once at the beginning of a session; it returns a Schema context which is used throughout the session.
2. The instance document to be validated is first parsed with the XML parser.
3. The XML context for the instance is then passed to the Schema validate function, along with an optional schema URL.
4. If no explicit schema is defined in the instance document, the default schema will be used.
5. More documents may then be validated using the same schema context.
6. When the session is over, the Schema tear down function is called, which releases all memory allocated by the loaded schemas.

**Figure 25–1 XMLSchema Processor for C Usage Diagram**

## How to Run XML Schema for C Sample Programs

This directory contains a sample XML Schema application that illustrates how to use Oracle XML Schema Processor with its API. [Table 25–3](#) lists the provided sample files.

**Table 25–3 XML Schema for C Samples Provided**

Sample File	Description
Makefile	Makefile to build the sample programs and run them, verifying correct output.
xsdtest.c	Trivial program which invokes the XML Schema for C API
car.{xsd,xml,std}	Sample schema, instance document, and expected output respectively, after running xsdtest on them. See: <ul style="list-style-type: none"> <li><a href="#">"XML Schema for C Example 2: car.xsd"</a> on page 25-11</li> <li><a href="#">"XML Schema for C Example 3: car.xml"</a> on page 25-12</li> <li><a href="#">"XML Schema for C Example 4: car.std"</a> on page 25-13.</li> </ul>

**Table 25–3 XML Schema for C Samples Provided**

Sample File	Description
aq.{xsd,xml,std}	Second sample schema, instance document, and expected output respectively, after running xsdtest on them. See: <a href="#">"XML Schema for C Example 5: aq.xsd"</a> on page 25-14 <a href="#">"XML Schema for C Example 6: aq.xml"</a> on page 25-23 <a href="#">"XML Schema for C Example 7: aq.std"</a> on page 25-24
pub.{xsd,xml,std}	Third sample schema, instance document, and expected output respectively, after running xsdtest on them. See: <a href="#">"XML Schema for C Example 8: pub.xsd"</a> on page 25-24 <a href="#">"XML Schema for C Example 9: pub.xml"</a> on page 25-26 <a href="#">"XML Schema for C Example 10: pub.std"</a> on page 25-27

To build the sample programs, run **'make'**.

To build the programs and run them, comparing the actual output to expected output, run **'make sure'**.

### Make.bat

```

:: #####
:: # Batch script to build Oracle XML parser C sample programs
:: #####
set opt_flg=-Ox -Oy-
if (%2) == (D) set opt_flg=-Z7 -Od
if (%2) == (D) set link_dbg=/debug /debugtype:both /pdb:none

if (%1) == () goto :XSDTEST
if (%1) == (all) goto :XSDTEST
if (%1) == (xsdtest) goto :XSDTEST
if (%1) == (clean) goto :CLEAN
if (%1) == (sure) goto :SURE
goto :EOF

:CLEAN
del *.obj
del *.out
del ..\bin\xsdtest.exe
goto :EOF

:XSDTEST

```

```

call :compile xsdtest
call :link xsdtest
if (%1) == (xsdtest) goto :EOF

:SURE
..\bin\xsdtest.exe car.xml > car.out
comp car.std car.out < NUL:
..\bin\xsdtest.exe pub.xml > pub.out
comp pub.std pub.out < NUL:
..\bin\xsdtest.exe aq.xml > aq.out
comp aq.std aq.out < NUL:
goto :EOF

:COMPILE
set filename=%1
cl -c -Fo%filename%.obj %opt_flg% /DCRTAPI1=_cdecl /DCRTAPI2=_cdecl /nologo /Zl
/Gy /DWIN32 /D_WIN32 /DWIN_NT /DWIN32COMMON /D_DLL /D_MT /D_X86_=1 -I.
-I..\include %filename%.c
goto :EOF

:LINK
set filename=%1
link %link_dbg% /out:..\bin\%filename%.exe /libpath:%ORACLE_HOME%\lib
/libpath:..\lib %filename%.obj oraxml8.lib oraxsd8.lib oracore8.lib oranls8.lib
user32.lib kernel32.lib msvcrt.lib ADVAPI32.lib oldnames.lib winmm.lib

:EOF

```

## XML Schema for C Example 1: xsdtest.c

```

/* Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved. */

/*
 NAME
 validate.c - Sample Schema validation

 DESCRIPTION
 Sample usage of C XML Schema processor
*/

#include <stdio.h>

#ifdef ORATYPES
include <oratypes.h>
#endif

```

```
#ifndef ORAXML_ORACLE
include <oraxml.h>
#endif

#ifndef ORAXSD_ORACLE
include <oraxsd.h>
#endif

int main(int argc, char **argv)
{
 xmlctx *ctx;
 xsdctx *sctx;
 char *doc, *schema;
 uword ecode;

 puts("XML C Schema processor");

 if ((argc < 2) || (argc > 3))
 {
 puts("usage: validate <xml document> [schema]");
 return -1;
 }
 doc = argv[1];
 schema = (argc > 2) ? argv[2] : 0;

 puts("Initializing XML package...");

 if (!(ctx = xmlinit(&ecode, (const oratext *) 0,
 (void (*)(void *, const oratext *, uword)) 0,
 (void *) 0, (const xmlsaxcb *) 0, (void *) 0,
 (const xmlmemcb *) 0, (void *) 0,
 (const oratext *) 0)))
 {
 printf("Failed to initialize XML parser, error %u\n", (unsigned) ecode);
 return 1;
 }

 printf("Parsing '%s' ...\n", doc);
 if (ecode = xmlparse(ctx, (oratext *) doc, (oratext *) 0,
XML_FLAG_DISCARD_WHITESPACE))
 {
 printf("Parse failed, error %u\n", (unsigned) ecode);
 return 2;
 }
}
```

```

puts("Initializing Schema package...");

if (!(scctx = schemaInitialize(ctx, &ecode))
 {
printf("Failed, code %u!\n", ecode);
return 3;
}

puts("Validating document...");
if (ecode = schemaValidate(scctx, ctx, (oratext *) schema))
{
printf("Validation failed, error %u\n", (unsigned) ecode);
return 4;
}

puts("Document is valid.");
return 0;
}

```

## XML Schema for C Example 2: car.xsd

```

<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
targetNamespace = "http://www.CarDealers.com/">
 <element name="Car">
<complexType>
 <element name="Model">
<simpleType base="string">
 <enumeration value = "Ford"/>
 <enumeration value = "Saab"/>
 <enumeration value = "Audi"/>
</simpleType>
 </element>
 <element name="Make">
<simpleType base="string">
 <minLength value = "1"/>
 <maxLength value = "30"/>
</simpleType>
</element>
 <element name="Year">
<complexType content="mixed">
 <attribute name="PreviouslyOwned" type="string"
use="required"/>
 <attribute name="YearsOwned" type="integer"

```

```
 use="optional"/>
</complexType>
</element>
<element name="OwnerName" type="string"
 minOccurs="0" maxOccurs="unbounded"/>
<element name="Condition">
<complexType base="string" derivedBy="extension">
 <attribute name="Automatic">
<simpleType base="string">
 <enumeration value = "Yes"/>
 <enumeration value = "No"/>
</simpleType>
</attribute>
</complexType>
</element>
<element name="Mileage">
<simpleType base="integer">
 <minInclusive value="0"/>
 <maxInclusive value="2000000"/>
</simpleType>
</element>
<attribute name="RequestDate" type="date"/>
</complexType>
</element>
</schema>
```

### XML Schema for C Example 3: car.xml

```
<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
 targetNamespace = "http://www.CarDealers.com/">
 <element name="Car">
<complexType>
 <element name="Model">
<simpleType base="string">
 <enumeration value = "Ford"/>
 <enumeration value = "Saab"/>
 <enumeration value = "Audi"/>
</simpleType>
</element>
 <element name="Make">
<simpleType base="string">
 <minLength value = "1"/>
 <maxLength value = "30"/>
```



```
</simpleType>
 </element>
 <element name="Year">
<complexType content="mixed">
 <attribute name="PreviouslyOwned" type="string"
 use="required" />
 <attribute name="YearsOwned" type="integer"
 use="optional" />
</complexType>
 </element>
 <element name="OwnerName" type="string"
 minOccurs="0" maxOccurs="unbounded" />
 <element name="Condition">
<complexType base="string" derivedBy="extension">
 <attribute name="Automatic">
<simpleType base="string">
 <enumeration value = "Yes" />
 <enumeration value = "No" />
</simpleType>
 </attribute>
</complexType>
 </element>
 <element name="Mileage">
<simpleType base="integer">
 <minInclusive value="0" />
 <maxInclusive value="2000000" />
</simpleType>
 </element>
 <attribute name="RequestDate" type="date" />
</complexType>
 </element>
</schema>
```

## XML Schema for C Example 4: car.std

```
XML C Schema processor
Initializing XML package...
Parsing 'car.xml' ...
Initializing Schema package...
Validating document...
Document is valid.
```

## XML Schema for C Example 5: aq.xsd

```
<?xml version="1.0"?>

<!-- ***** AQ xml schema ***** -->

<schema xmlns = "http://www.w3.org/1999/XMLSchema"
 targetNamespace = "http://www.oracle.com/AQXmlDocument"
 xmlns:aq = "http://www.oracle.com/AQXmlDocument"
 xmlns:xsd = "http://www.w3.org/1999/XMLSchema"
 elementFormDefault="qualified">

 <element name="AQXmlDocument">
 <complexType content="mixed">
 <choice>
 <group ref="aq:client_operation" minOccurs="0"/>
 <group ref="aq:server_response"/>
 </choice>
 </complexType>
 </element>

 <!-- ***** Client Operations Group ***** -->
 <group name="client_operation">
 <sequence>
 <element ref="aq:client_operation" minOccurs="0" maxOccurs="1"/>
 <choice>
 <element ref="aq:producer_options" maxOccurs="1"/>
 <element ref="aq:consumer_options" maxOccurs="1"/>
 <element ref="aq:register_options" maxOccurs="1"/>
 </choice>
 <element ref="aq:message_set" minOccurs="0" maxOccurs="*/>
 </sequence>
 </group>

 <!-- ***** Server Response Group ***** -->
 <group name="server_response">
 <sequence>
 <element ref="aq:server_response" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:receive_result" maxOccurs="1"/>
 <choice minOccurs="0">
 <element ref="aq:send_result" maxOccurs="1"/>
 <element ref="aq:publish_result" maxOccurs="1"/>
 <element ref="aq:receive_result" maxOccurs="1"/>
 <element ref="aq:sequence_num_result" maxOccurs="1"/>
 </choice>
 </sequence>
 </group>
</schema>
```

```

 </choice>
 </sequence>
</group>

<!-- ***** Server Propagation Group ***** -->
<group name="server_prop_operation">
 <sequence>
 <element ref="aq:server_prop_operation" minOccurs="0" maxOccurs="1"/>
 <choice>
 <element ref="aq:push" maxOccurs="1"/>
 <element ref="aq:notification" maxOccurs="1"/>
 <element ref="aq:sequence_num_request" maxOccurs="1"/>
 </choice>
 </sequence>
</group>

<!-- ***** Client Operation ***** -->
<element name="client_operation">
 <complexType content="mixed">
 <element ref="aq:txid" minOccurs="0"/>
<attribute name="opcode" use="required" type="aq:opcode_type"/>
 </complexType>
</element>

<!-- ***** Server Response ***** -->
<element name="server_response">
 <complexType content="mixed">
 <element ref="aq:txid" minOccurs="0"/>
 <element ref="aq:status_response" minOccurs="1"/>
<attribute name="opcode" use="required" type="aq:opcode_type"/>
 </complexType>
</element>

<!-- ***** Server Propagation Operation ***** -->
<element name="server_prop_operation">
 <complexType content="mixed">
 <element ref="aq:txid" minOccurs="0"/>
<attribute name="prop_opcode" use="required" type="aq:prop_opcode_type"/>
 </complexType>
</element>

<element name="txid" type="string"/>

<element name="destination">
 <complexType base='string' derivedBy="extension">

```

```

 <attribute name="lookup_type" type="aq:dest_lookup_type"
 use="default" value="NORMAL"/>
 </complexType>
</element>

<!-- **** destination lookup type ***** -->
<!-- lookup_type can be specified to either lookup LDAP or use -->
<!-- the destination directly in NAME::ADDRESS::PROTOCOL format -->
<simpleType name="dest_lookup_type" base="string">
 <enumeration value="NORMAL"/>
 <enumeration value="LDAP"/>
</simpleType>

<!-- ***** Producer Options ***** -->
<element name="producer_options">
 <complexType content="mixed">
 <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:priority" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:expiration" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:recipient_list" minOccurs="0" maxOccurs="1"/>
 <attribute name="visibility" type="aq:visibility_type"
 use="default" value="ON_COMMIT"/>
 <attribute name="delivery_mode" type="aq:del_mode_type"
 use="default" value="PERSISTENT"/>
 </complexType>
</element>

<!-- ***** Consumer Options ***** -->
<element name="consumer_options">
 <complexType content="mixed">
 <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:consumer_name" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:wait_time" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:selector" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:batch_size" minOccurs="0" maxOccurs="1"/>
 <attribute name="visibility" type="aq:visibility_type"
 use="default" value="ON_COMMIT"/>
 <attribute name="dequeue_mode" type="aq:deq_mode_type"
 use="default" value="REMOVE"/>
 <attribute name="navigation" type="aq:nav_mode_type"
 use="default" value="NEXT_MESSAGE"/>
 </complexType>
</element>

```

```

<!-- ***** Register Options ***** -->
<element name="register_options">
 <complexType content="mixed">
 <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:consumer_name" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:notify_url" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:qos" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:batch_size" minOccurs="0" maxOccurs="1"/>
 </complexType>
</element>

<element name="recipient_list">
 <complexType content="mixed">
<element ref="aq:recipient" minOccurs="1" maxOccurs="*" />
 </complexType>
</element>

<!-- ***** Message Set ***** -->
<element name="message_set">
 <complexType content="mixed">
 <element ref="aq:message_count" minOccurs="1"/>
 <element ref="aq:message" minOccurs="0" maxOccurs="*" />
 </complexType>
</element>

<!-- ***** Message ***** -->
<element name="message">
 <complexType content="mixed">
 <element ref="aq:message_number" minOccurs="0"/>
 <element ref="aq:message_header" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:message_payload" minOccurs="0" maxOccurs="1"/>
 </complexType>
</element>

<!-- ***** Message header ***** -->
<element name="message_header">
 <complexType content="mixed">
 <element ref="aq:message_id" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:correlation" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:delay" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:priority" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:delivery_count" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:message_state" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:sender_id" minOccurs="1" maxOccurs="1"/>
 </complexType>
</element>

```

```

 <element ref="aq:exception_queue" minOccurs="0" maxOccurs="1"/>
 </complexType>
</element>

<!-- ***** Oracle JMS properties ***** -->
<element name="oracle_jms_properties">
 <complexType content="mixed">
 <element ref="aq:type" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:reply_to" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:userid" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:appid" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:groupid" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:group_sequence" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:timestamp" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:recv_timestamp" minOccurs="0" maxOccurs="1"/>
 </complexType>
</element>

<!-- ***** Message payload ***** -->
<element name="message_payload">
 <complexType>
 <choice>
 <element ref="aq:jms_text_message" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:jms_map_message" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:jms_bytes_message" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:jms_object_message" minOccurs="0" maxOccurs="1"/>
 <any minOccurs="0" maxOccurs="*" processContents="skip"/>
 </choice>
 </complexType>
</element>

...

<!-- ***** Status response ***** -->
<element name="status_response">
 <complexType content="mixed">
 <element ref="aq:acknowledge" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:status_code" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:error_code" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:error_message" minOccurs="0" maxOccurs="1"/>
 </complexType>
</element>

<!-- ***** Send result ***** -->
<element name="send_result">
 <complexType content="mixed">

```

```

 <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:message_id" minOccurs="0" maxOccurs="*/>
 </complexType>
</element>

<!-- ***** Publish result ***** -->
<element name="publish_result">
 <complexType content="mixed">
 <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:message_id" minOccurs="0" maxOccurs="*/>
 </complexType>
</element>

<!-- ***** Receive result ***** -->
<element name="receive_result">
 <complexType content="mixed">
 <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:message_set" minOccurs="0" maxOccurs="*/>
 </complexType>
</element>

....

<!-- ***** Push messages ***** -->
<element name="push">
 <complexType content="mixed">
 <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:consumer_name" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:sequence_number" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:message_set" minOccurs="1" maxOccurs="1"/>
 </complexType>
</element>

<!-- ***** Notification ***** -->
<element name="notification">
 <complexType content="mixed">
 <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:consumer_name" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:message" minOccurs="0" maxOccurs="1"/>
 </complexType>
</element>

<element name="priority" type="integer"/>
<element name="expiration" type="integer"/>
<element name="consumer_name" type="string"/>
<element name="wait_time" type="integer"/>

```

```
<element name="batch_size" type="integer"/>
<element name="qos" type="string"/>
<element name="notify_url" type="string"/>
<element name="message_id" type="string"/>
<element name="message_state" type="string"/>

<element name="sequence_number" type="integer"/>
<element name="message_number" type="integer"/>
<element name="message_count" type="integer"/>

<element name="correlation" type="string"/>
<element name="delay" type="integer"/>
<element name="delivery_count" type="integer"/>
<element name="exception_queue" type="string"/>

<element name="type" type="string"/>
<element name="userid" type="string"/>
<element name="appid" type="string"/>
<element name="groupid" type="string"/>
<element name="group_sequence" type="integer"/>
<element name="timestamp" type="date"/>
<element name="recv_timestamp" type="date"/>

<element name="recipient">
 <complexType base='string' derivedBy="extension">
 <attribute name="lookup_type" type="aq:dest_lookup_type"
 use="default" value="NORMAL"/>
 </complexType>
</element>

<element name="sender_id">
 <complexType base='string' derivedBy="extension">
 <attribute name="lookup_type" type="aq:dest_lookup_type"
 use="default" value="NORMAL"/>
 </complexType>
</element>

<element name="reply_to">
 <complexType base='string' derivedBy="extension">
 <attribute name="lookup_type" type="aq:dest_lookup_type"
 use="default" value="NORMAL"/>
 </complexType>
</element>

<element name="selector">
```



```
<complexType>
<choice>
 <element ref="aq:correlation_id" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:message_id" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:complex_selector" minOccurs="0" maxOccurs="1"/>
</choice>
</complexType>
</element>

<element name="correlation_id" type="string"/>
<element name="complex_selector" type="string"/>

<simpleType name="visibility_type" base="string">
 <enumeration value="ON_COMMIT"/>
 <enumeration value="IMMEDIATE"/>
</simpleType>

<simpleType name="del_mode_type" base="string">
 <enumeration value="PERSISTENT"/>
 <enumeration value="NONPERSISTENT"/>
</simpleType>

<simpleType name="deq_mode_type" base="string">
 <enumeration value="BROWSE"/>
 <enumeration value="LOCKED"/>
 <enumeration value="REMOVE"/>
 <enumeration value="REMOVE_NO_DATA"/>
</simpleType>

<simpleType name="nav_mode_type" base="string">
 <enumeration value="FIRST_MESSAGE"/>
 <enumeration value="NEXT_MESSAGE"/>
 <enumeration value="NEXT_TRANSACTION"/>
</simpleType>

<simpleType name="opcode_type" base="string">
 <enumeration value="SEND"/>
 <enumeration value="RECEIVE"/>
 <enumeration value="PUBLISH"/>
 <enumeration value="REGISTER"/>
 <enumeration value="COMMIT"/>
 <enumeration value="ROLLBACK"/>
 <enumeration value="SEQ_NUM_REQUEST"/>
</simpleType>
```

```

<simpleType name="prop_opcode_type" base="string">
 <enumeration value="SEND"/>
 <enumeration value="NOTIFICATION"/>
 <enumeration value="COMMIT"/>
 <enumeration value="ROLLBACK"/>
 <enumeration value="SEQ_NUM_REQUEST"/>
</simpleType>

<element name="acknowledge">
 <complexType content="empty">
 </complexType>
</element>
<element name="status_code" type="string"/>
<element name="error_code" type="string"/>
<element name="error_message" type="string"/>

<simpleType name="prop_type" base="string">
 <enumeration value="STRING"/>
 <enumeration value="NUMBER"/>
</simpleType>

<element name="name" type="string"/>
<element name="value" type="string"/>

<!-- ***** JMS text message ***** -->
<element name="jms_text_message">
 <complexType content="mixed">
 <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:text_data" minOccurs="1" maxOccurs="1"/>
 </complexType>
</element>
<element name="text_data" type="string"/>

....

<!-- ***** JMS object message ***** -->
<element name="jms_object_message">
 <complexType content="mixed">
 <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:user_properties" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:ser_object_data" minOccurs="1" maxOccurs="1"/>
 </complexType>
</element>

```

```

 <element name="ser_object_data" type="string"/>
</schema>

```

## XML Schema for C Example 6: aq.xml

```

<AQXMLDocument xmlns="http://www.oracle.com/AQXMLDocument"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xsi:schemaLocation="http://www.oracle.com/AQXMLDocument aq.xsd">

 <client_operation opcode="SEND">
 <txid> sdasdfsdf </txid>
 </client_operation>

 <producer_options delivery_mode="PERSISTENT">
 <destination lookup_type="NORMAL"> queue1 </destination>
 <priority>23</priority>
 <recipient_list>
 <recipient> abc </recipient>
 <recipient lookup_type="LDAP"> abc </recipient>
 </recipient_list>
 </producer_options>

 <message_set>
 <message_count>1</message_count>
 <message>
 <message_number>1</message_number>
 <message_header>
 <correlation>XML_40_NEW_TEST</correlation>
 <delay>10</delay>
 <sender_id>scott::home::0</sender_id>
 </message_header>
 <message_payload>
 <jms_map_message>
 <oracle_jms_properties>
 <reply_to>oracle::redwoodshores::100</reply_to>
 <userid>scott</userid>
 <appid>AQProduct</appid>
 <groupid>AQ</groupid>
 </oracle_jms_properties>
 <user_properties>
 <property property_type="STRING">
 <name>country</name>
 <value>USA</value>
 </property>
 <property property_type="STRING">

```

```
 <name>State</name>
 <value>california</value>
 </property>
</user_properties>
<map_data>
 <item item_type="STRING">
 <name>Car</name>
 <value>Toyota</value>
 </item>
 <item item_type="STRING">
 <name>Color</name>
 <value>Blue</value>
 </item>
 <item item_type="STRING">
 <name>Shape</name>
 <value>Circle</value>
 </item>
 <item item_type="NUMBER">
 <name>Price</name>
 <value>20000</value>
 </item>
</map_data>
</jms_map_message>
</message_payload>
</message>
</message_set>
</AQXmlDocument>
```

### XML Schema for C Example 7: aq.std

```
XML C Schema processor
Initializing XML package...
Parsing 'aq.xml' ...
Initializing Schema package...
Validating document...
Document is valid.
```

### XML Schema for C Example 8: pub.xsd

```
<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/2000/08/XMLSchema"
 targetNamespace = "http://www.somewhere.org/BookCatalogue"
 xmlns:cat = "http://www.somewhere.org/BookCatalogue"
 elementFormDefault="qualified">
```

```

<complexType name="Pub">
 <sequence>
 <element name="Title" type="cat:titleType" maxOccurs="*" />
 <element name="Author" type="string" maxOccurs="*" />
 <element name="Date" type="date" />
 </sequence>
 <attribute name="language" type="string" use="default" value="English" />
 <anyAttribute namespace="##local" />
</complexType>

<element name="Publication" type="cat:Pub" abstract="true" />
<element name="Book" substitutionGroup="cat:Publication">
 <complexType>
 <complexContent>
 <extension base="cat:Pub" >
 <sequence>
 <element name="ISBN" type="string" default="123456789" />
 <element name="Publisher" type="string" />
 </sequence>
 </extension>
 </complexContent>
 </complexType>
</element>
<complexType name="titleType">
 <simpleContent>
 <extension base="string" >
 <attribute name="old" type="string" use="default" value="false" />
 </extension>
 </simpleContent>
</complexType>
<element name="Magazine" substitutionGroup="cat:Publication">
 <complexType>
 <complexContent>
 <extension base="cat:Pub">
 <sequence>
 <element name="Volume" type="cat:VolumeType" />
 <element name="htmlTable">
 <complexType>
 <any namespace="##other"
 processContents="skip"
 minOccurs="0" maxOccurs="2" />
 </complexType>
 </element>
 </sequence>
 </extension>
 </complexContent>
 </complexType>

```

```

 </complexContent>
 </complexType>
</element>
<simpleType name="VolumeType">
 <restriction base="integer" >
 <minInclusive value = "1"/>
 <maxInclusive value = "12"/>
 </restriction>
</simpleType>
<element name="Catalogue">
 <complexType>
 <sequence>
 <element ref="cat:Publication" minOccurs="0" maxOccurs="*" />
 </sequence>
 </complexType>
</element>
</schema>

```

## XML Schema for C Example 9: pub.xml

```

<?xml version="1.0"?>
<?xml version="1.0"?>

<Catalogue xmlns = "http://www.somewhere.org/BookCatalogue"
 xmlns:cat = "http://www.somewhere.org/BookCatalogue"
 xmlns:html = "http://www.somewhere.org/HTMLCatalogue"
 xmlns:xsi = "http://www.w3.org/1999/XMLSchema-instance"
 xsi:schemaLocation =
 "http://www.somewhere.org/BookCatalogue pub.xsd">

 <cat:Magazine>
 <Title>Natural Health</Title>
 <Author>October</Author>
 <Date>1999-12</Date>
 <Volume>12</Volume>
 <htmlTable>
 <table xmlns = "http://www.somewhere.org/HTMLCatalogue">
 <tr>...</tr>
 </table>
 <html:table>
 <html:tr>...</html:tr>
 </html:table>
 </htmlTable>
 </cat:Magazine>

```

```
<Book>
 <Title>Illusions The Adventures of a Reluctant Messiah</Title>
 <Author>Richard Bach</Author>
 <Date>1977</Date>
 <ISBN></ISBN>
 <Publisher>Dell Publishing Co.</Publisher>
</Book>

<Book>
 <Title>The First and Last Freedom</Title>
 <Author>J. Krishnamurti</Author>
 <Date>1954</Date>
 <ISBN>0-06-064831-7</ISBN>
 <Publisher>Harper & Row</Publisher>
</Book>

</Catalogue>
```

## XML Schema for C Example 10: pub.std

```
XML C Schema processor
Initializing XML package...
Parsing 'pub.xml' ...
Initializing Schema package...
Validating document...
Document is valid.
```





# Part IX

---

## XDK for C++

Part IX describes how to access and use Oracle's XML Developer's Kit (XDK) for C++. It contains the following chapters:

- [Chapter 26, "Using XML Parser for C++"](#)
- [Chapter 27, "Using XML Schema Processor for C++"](#)
- [Chapter 28, "Using XML C++ Class Generator"](#)



---

## Using XML Parser for C++

This chapter contains the following sections:

- [Accessing XML Parser for C++](#)
- [XML Parser for C++ Features](#)
- [XML Parser for C++ Usage](#)
- [XML Parser for C++ XSLT \(DOM Interface\) Usage](#)
- [Default Behavior](#)
- [DOM and SAX APIs](#)
- [Invoking XML Parser for C++](#)
- [Using the Sample Files Included with Your Software](#)
- [Running the XML Parser for C++ Sample Programs](#)

## Accessing XML Parser for C++

The XML Parser for C++ is provided with Oracle and is also available for download from the OTN site: <http://otn.oracle.com/tech/xml>.

It is located at `$ORACLE_HOME/xdk/cpp/parser`.

## XML Parser for C++ Features

`readme.html` in the root directory of the software archive contains release specific information including bug fixes and API additions.

XML Parser for C++ will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface.

You can post questions, comments, or bug reports to the XML Discussion Forum at <http://otn.oracle.com>.

## Specifications

See [Appendix F, "XDK for C++: Specifications and Cheat Sheet"](#) for a list of XML Parser for C++ specifications and methods.

### See Also:

- The doc directory in your install area
- *Oracle9i XML Reference*

## Memory Allocation

The memory callback functions `memcb` may be used if you wish to use your own memory allocation. If they are used, all of the functions should be specified.

The 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:

- `xmlparse()` or `xmlparsebuf()` is called to parse another file or buffer.
- `xmlclean()` is called.
- `xmlterm()` is called.

## Thread Safety

If threads are forked off somewhere in the midst of the init-parse-term sequence of calls, you will get unpredictable behavior and results.

## Data Types Index

[Table 26–1](#) lists the datatypes used in XML Parser for C++.

**Table 26–1** *Datatypes Used in XML Parser for C++*

Data Type	Description
oratext	String pointer
xmlctx	Master XML context
xmlmemcb	Memory callback structure (optional)
xmlsaxcb	SAX callback structure (SAX only)
ub4	32-bit (or larger) unsigned integer
uword	Native unsigned integer

## Error Message Files

Error message files are provided in the `mesg/` subdirectory. The messages files also exist in the `$ORACLE_HOME/oracore/mesg` directory. You may set the environment variable `ORA_XML_MESG` to point to the absolute path of the `mesg/` subdirectory although this not required.

## Validation Modes

Available validation modes are described in [Chapter 20, "Using XML Parser for Java"](#), ["Oracle XML Parsers Support Four Validation Modes"](#) on page 20-5.

## XML Parser for C++ Usage

[Figure 26–1](#) illustrates the XML Parser for C++ functionality.

1. The parsing process begins with the `xmlinit()` method.
2. The XML input can be either an XML file or string buffer. This inputs the following methods:

- `XMLParser.xmlparse()` if the input is an XML file
- `XMLParser.xmlparseBuffer()` if the input is a string buffer

### 3. DOM or SAX API

**DOM:** If you are using the DOM interface, include the following steps:

- The `XMLParser.xmlparse()` or `xmlparseBuffer()` method calls `.getDocumentElement()`. If no other DOM methods are being applied, you can invoke `.xmlterm()`.
- This optionally calls other DOM methods if required. These are typically Node class methods or print methods. It outputs the DOM document.
- If complete, the process invokes `.xmlterm()`
- You can optionally first invoke `.xmlclean()` to clean up any data structure created during the parse process. You would then call `.xmlterm()`

**SAX:** If you are using the SAX interface, include the following steps:

- Process the results of the parser from `.xmlparse()` or `.xmlparseBuffer()` via callback methods.
  - Register the callback methods
4. Optionally, use `.xmlclean()` to clean up the memory and structures used during a parse, and go to Step 5. or return to Step 2.
  5. Terminate the parsing process with `.xmlterm()`

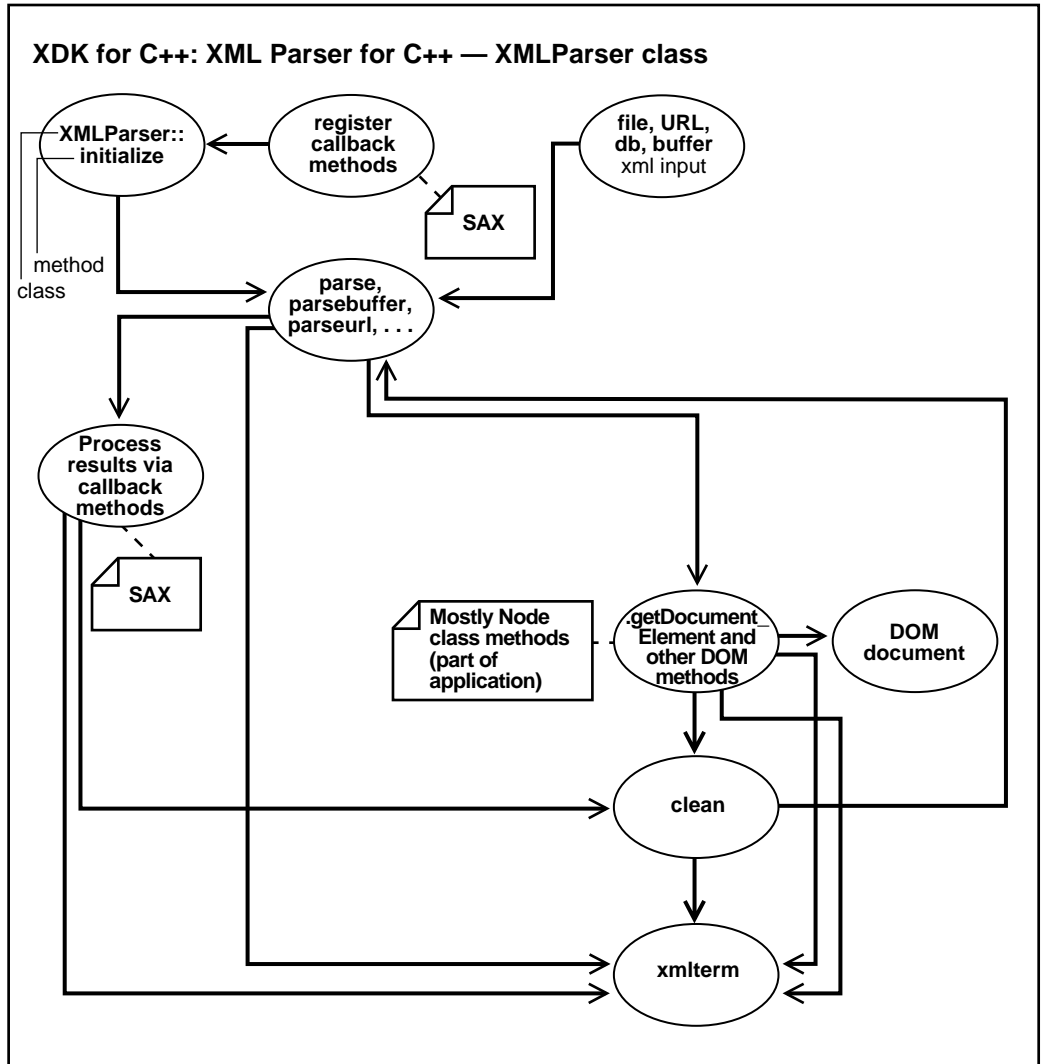
### Parser Calling Sequence

The sequence of calls to the parser can be any of the following:

- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` or `XMLParser.xmlparsebuf()` - `XMLParser.xmlterm()`
- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` or `XMLParser.xmlparsebuf()` - `XMLParser.xmlclean()` - `XMLParser.xmlparse()` or `XMLParser.xmlparsebuf()` - `XMLParser.xmlclean()` - ... - `XMLParser.xmlterm()`
- `XMLParser.xmlinit()` - `XMLParser.xmlparse()` or

```
XMLParser.xmlparsebuf() - XMLParser.xmlparse() or
XMLParser.xmlparsebuf() - ... - XMLParser.xmlterm()
```

Figure 26-1 XML Parser for C++ (DOM and SAX Interfaces) Usage



## XML Parser for C++ XSLT (DOM Interface) Usage

Figure 26-2 shows the XML Parser for C++ XSLT functionality for the DOM interface.

1. There are two inputs to `XMLParser.xmlparse()`:
  - The Stylesheet to be applied to the XML document
  - XML document

The output of `XMLParser.xmlparse()`, the parsed stylesheet and parsed XML document, are sent to the `XSLProcess.xslprocess()` method for processing.

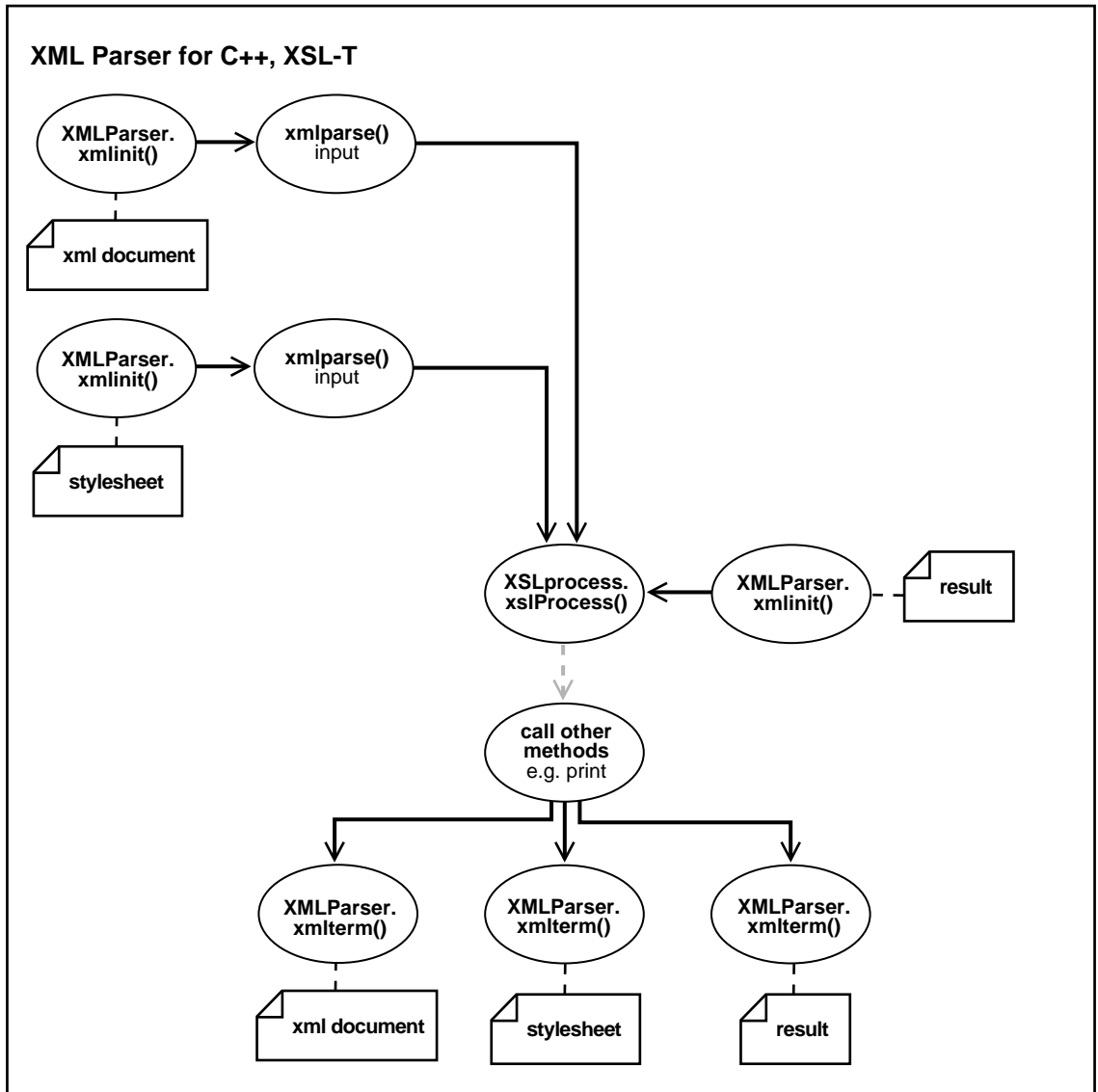
2. `XMLParser.xmlinit()` initializes the XSLT processing. `XMLParser.xmlinit()` also initializes the `xslprocess()` result
3. `XSLProcess.xslProcess()` optionally calls other methods, such as print methods. You can see the list of available methods either on OTN or in *Oracle9i XML Reference*.
4. The resultant document (XML, HTML, VML, and so on) is typically sent to an application for further processing.
5. The application terminates the XSLT process by declaring `XMLParser.xmlterm()`, for the XML document, stylesheet, and final result.

XML Parser for C XSLT functionality is illustrated with the following examples:

- [XML Parser for C++ Example 16: C++ — XSLSample.cpp](#) on page 26-52
- [XML Parser for C++ Example 17: C++ — XSLSample.std](#) on page 26-54



Figure 26-2 Parser for C++: XSL-T Functionality (DOM Interface) Usage



## Default Behavior

The following is the XML Parser for C++ default behavior:

- Character set encoding is UTF-8. If all your documents are ASCII, you are encouraged to set the encoding to US-ASCII for better performance.
- Messages are printed to stderr unless msghdlr is given.
- A parse tree which can be accessed by DOM APIs is built unless saxcb is set to use the SAX callback APIs. Note that any of the SAX callback functions can be set to NULL if not needed.
- The default behavior for the parser is to check that the input is well-formed but not to check whether it is valid. The flag `XML_FLAG_VALIDATE` can be set to validate the input. The default behavior for whitespace processing is to be fully conformant to the XML 1.0 spec, that is, all whitespace is reported back to the application but it is indicated which whitespace is ignorable. However, some applications may prefer to set the `XML_FLAG_DISCARD_WHITESPACE` which will discard all whitespace between an end-element tag and the following start-element tag.

---

---

**Note:** It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

---

---

## DOM and SAX APIs

Oracle XML parser for C++ checks if an XML document is well-formed, and optionally validates it against a DTD. The parser constructs an object tree which can be accessed via one of the following interfaces:

- DOM interface
- Serially via a SAX interface

These two XML APIs:

- **DOM: Tree-based APIs.** A tree-based API compiles an XML document into an internal tree structure, then allows an application to navigate that tree using the Document Object Model (DOM), a standard tree-based API for XML and HTML documents.
- **SAX: Event-based APIs.** An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface.

Tree-based APIs are useful for a wide range of applications, but they often put a great strain on system resources, especially if the document is large (under very controlled circumstances, it is possible to construct the tree in a lazy fashion to avoid some of this problem). Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

In both of these cases, an event-based API provides a simpler, lower-level access to an XML document: you can parse documents much larger than your available system memory, and you can construct your own data structures using your callback event handlers.

## Using the SAX API

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 can also be included. That context pointer will be passed to each SAX function.

### SAX Callback Structure

The SAX callback structure:

```
typedef struct
```

```

{
 sword (*startDocument)(void *ctx);
 sword (*endDocument)(void *ctx);
 sword (*startElement)(void *ctx, const oratext *name,
 const struct xmlarray *attrs);
 sword (*endElement)(void *ctx, const oratext *name);
 sword (*characters)(void *ctx, const oratext *ch, size_t len);
 sword (*ignorableWhitespace)(void *ctx, const oratext *ch, size_t len);
 sword (*processingInstruction)(void *ctx, const oratext *target,
 const oratext *data);
 sword (*notationDecl)(void *ctx, const oratext *name,
 const oratext *publicId, const oratext *systemId);
 sword (*unparsedEntityDecl)(void *ctx, const oratext *name,
 const oratext *publicId,
 const oratext *systemId, const oratext *notationName);
 sword (*nsStartElement)(void *ctx, const oratext *qname,
 const oratext *local, const oratext *nsp,
 const struct xmlnodes *attrs);
} xmlsaxcb;

```

## Using the DOM API

See "[XML Parser for C++ Example 6: C++ — DOMSample.cpp](#)" on page 26-16.

## Invoking XML Parser for C++

XML Parser for C++ can be invoked in two ways:

- By invoking the executable on the command line
- By writing C++ code and using the supplied APIs

## Command Line Usage

The XML Parser for C++ can be called as an executable by invoking `bin/xml`

[Table 26–2](#) lists the command line options.

**Table 26–2 XML Parser for C++: Command Line Options**

Option	Description
-c	Conformance check only, no validation
-e encoding	Specify input file encoding

**Table 26–2 XML Parser for C++: Command Line Options**

Option	Description
-h	Help - show this usage help
-n	Number - DOM traverse and report number of elements
-p	Print document and DTD structures after parse
-x	Exercise SAX interface and print document
-v	Version - display parser version then exit
-w	Whitespace - preserve all whitespace

## Writing C++ Code to Use Supplied APIs

XML Parser for C++ can also be invoked by writing code to use the supplied APIs. The code must be compiled using the headers in the `include/` subdirectory and linked against the libraries in the `lib/` subdirectory. Please see the `Makefile` in the `sample/` subdirectory for full details of how to build your program.

## Using the Sample Files Included with Your Software

`$ORACLE_HOME/xdk/cpp/parser/sample/` directory contains several XML applications to illustrate how to use the XML Parser for C++ with the DOM and SAX interfaces.

[Table 26–3](#) lists the sample files in `sample/` directory.

**Table 26–3 XML Parser for C++ sample/ Files**

sample/ File Name	Description
<code>DOMNamespace.cpp</code>	Source for <code>DOMNamespace</code> program
<code>DOMNamespace.std</code>	Expected output from <code>DOMNamespace</code>
<code>DOMSample.cpp</code>	Source for <code>DOMSample</code> program
<code>DOMSample.std</code>	Expected output from <code>DOMSample</code>
<code>FullDOM.c</code>	Sample usage of DOM interface
<code>FullDOM.std</code>	Expected output from <code>FullDOM</code>
<code>Make.bat</code>	Batch file to build sample executables
<code>Makefile</code>	Makefile for sample programs

**Table 26–3 XML Parser for C++ sample/ Files(Cont.)**

<b>sample/ File Name</b>	<b>Description</b>
NSExample.xml	Sample XML file using namespaces
SAXNamespace.cpp	Source for SAXNamespace program
SAXNamespace.std	Expected output from SAXNamespace
SAXSample.cpp	Source for SAXSample program
SAXSample.std	Expected output from SAXSample
XSLSample.cpp	Source for XSLSample program
XSLSample.std	Expected output from XSLSample
class.xml	XML file that may be used with XSLSample
iden.xsl	Stylesheet that may be used with XSLSample
cleo.xml	XML version of Shakespeare's play

## Running the XML Parser for C++ Sample Programs

### Building the Sample programs

Change directories to `..sample/` and read the README file. This will explain how to build the sample programs according to your platform.

### Sample Programs

[Table 26–4](#) lists the programs built by the sample files in `sample/`.

**Table 26–4 XML Parser for C++, Sample Programs Built in sample/**

<b>Built Program</b>	<b>Description</b>
SAXSample	A sample application using SAX APIs. Prints out all speakers in each scene, i.e. all the unique SPEAKER elements within each SCENE element.
DOMSample [speaker]	A sample application using DOM APIs. Prints all speeches made by the given speaker. If no speaker is specified, "Soothsayer" is used. Note that major characters have uppercase names (e.g. "CLEOPATRA"), whereas minor characters have capitalized names (e.g. "Attendant"). See the output of SAXSample.

**Table 26–4 XML Parser for C++, Sample Programs Built in sample/**

Built Program	Description
SAXNamespace	A sample application using Namespace extensions to SAX API; prints out all elements and attributes of NSExample.xml along with full namespace information.
DOMNamespace	Same as SAXNamespace except using DOM interface.
FullDOM	Sample usage of full DOM interface. Exercises all the calls, but does nothing too exciting.
XSLSample <xmlfile> <xsl ss>	Sample usage of XSL processor. It takes two filenames as input, the XML file and the XSL stylesheet. Note: If you redirect stdout of this program to a file, you may encounter some missing output, depending on your environment.

## XML Parser for C++ Example 1: XML — class.xml

class.xml is an XML file that inputs XSLSample.cpp.

```
<?xml version = "1.0"?>
<!DOCTYPE course [
<!ELEMENT course (Name, Dept, Instructor, Student)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Dept (#PCDATA)>
<!ELEMENT Instructor (Name)>
<!ELEMENT Student (Name*)>
]>
<course>
<Name>Calculus</Name>
<Dept>Math</Dept>
<Instructor>
<Name>Jim Green</Name>
</Instructor>
<Student>
<Name>Jack</Name>
<Name>Mary</Name>
<Name>Paul</Name>
</Student>
</course>
```

## XML Parser for C++ Example 2: XML — cleo.xml

This XML example inputs DOMSample.cpp and SAXSample.cpp.

```
<?xml version="1.0"?>
<!DOCTYPE PLAY [
 <!ELEMENT PLAY (TITLE, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
 PROLOGUE?, ACT+, EPILOGUE?)>
 <!ELEMENT TITLE (#PCDATA)>
 <!ELEMENT FM (P+)>
 <!ELEMENT P (#PCDATA)>
 <!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
 <!ELEMENT PGROUP (PERSONA+, GRPDESCR)>
 <!ELEMENT PERSONA (#PCDATA)>
 <!ELEMENT GRPDESCR (#PCDATA)>
 <!ELEMENT SCNDESCR (#PCDATA)>
 <!ELEMENT PLAYSUBT (#PCDATA)>
 <!ELEMENT INDUCT (TITLE, SUBTITLE*, (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
 <!ELEMENT ACT (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
 <!ELEMENT SCENE (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
 <!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
 <!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
 <!ELEMENT SPEECH (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
 <!ELEMENT SPEAKER (#PCDATA)>
 <!ELEMENT LINE (#PCDATA | STAGEDIR)*>
 <!ELEMENT STAGEDIR (#PCDATA)>
 <!ELEMENT SUBTITLE (#PCDATA)>
 <!ELEMENT SUBHEAD (#PCDATA)>
]>

<PLAY>
<TITLE>The Tragedy of Antony and Cleopatra</TITLE>

<PERSONAE>
<TITLE>Dramatis Personae</TITLE>

<PGROUP>
<PERSONA>MARK ANTONY</PERSONA>
<PERSONA>OCTAVIUS CAESAR</PERSONA>
<PERSONA>M. AEMILIUS LEPIDUS</PERSONA>
<GRPDESCR>triumvirs.</GRPDESCR>
</PGROUP>

<PERSONA>SEXTUS POMPEIUS</PERSONA>

<PGROUP>
```



```
<PERSONA>DOMITIUS ENOBARBUS</PERSONA>
<PERSONA>VENTIDIUS</PERSONA>
<PERSONA>EROS</PERSONA>
<PERSONA>SCARUS</PERSONA>
<PERSONA>DERCETAS</PERSONA>
<PERSONA>DEMETRIUS</PERSONA>
<PERSONA>PHILO</PERSONA>
<GRPDESCR>friends to Antony.</GRPDESCR>
</PGROUP>

<PGROUP>
<PERSONA>MECAENAS</PERSONA>
<PERSONA>AGRIPPA</PERSONA>
<PERSONA>DOLABELLA</PERSONA>
<PERSONA>PROCULEIUS</PERSONA>
<PERSONA>THYREUS</PERSONA>
<PERSONA>GALLUS</PERSONA>
<PERSONA>MENAS</PERSONA>
<GRPDESCR>friends to Caesar.</GRPDESCR>
</PGROUP>
...
...
<SPEECH>
<SPEAKER>First Guard</SPEAKER>
<LINE>This is an aspic's trail: and these fig-leaves</LINE>
<LINE>Have slime upon them, such as the aspic leaves</LINE>
<LINE>Upon the caves of Nile.</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>OCTAVIUS CAESAR</SPEAKER>
<LINE>Most probable</LINE>
<LINE>That so she died; for her physician tells me</LINE>
<LINE>She hath pursued conclusions infinite</LINE>
<LINE>Of easy ways to die. Take up her bed;</LINE>
<LINE>And bear her women from the monument:</LINE>
<LINE>She shall be buried by her Antony:</LINE>
<LINE>No grave upon the earth shall clip in it</LINE>
<LINE>A pair so famous. High events as these</LINE>
<LINE>Strike those that make them; and their story is</LINE>
<LINE>No less in pity than his glory which</LINE>
<LINE>Brought them to be lamented. Our army shall</LINE>
<LINE>In solemn show attend this funeral;</LINE>
<LINE>And then to Rome. Come, Dolabella, see</LINE>
<LINE>High order in this great solemnity.</LINE>
```

```

</SPEECH>

<STAGEDIR>Exeunt</STAGEDIR>
</SCENE>
</ACT>
</PLAY>

```

## XML Parser for C++ Example 3: XSL — iden.xsl

This example stylesheet can be used to input `XSLSample.cpp`.

```

<?xml version="1.0"?>
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <xsl:template match="*|*|comment()|processing-instruction()|text()">
 <xsl:copy>
 <xsl:apply-templates
select="*|*|comment()|processing-instruction()|text()"/>
 </xsl:copy>
 </xsl:template>

</xsl:stylesheet>

```

## XML Parser for C++ Example 4: XML — FullDOM.xml (DTD)

This example DTD file inputs `FullDOM.cpp`.

```

<!DOCTYPE doc [
 <!ELEMENT p (#PCDATA)>
 <!ATTLIST p xml:space (preserve|default) 'preserve'>
 <!NOTATION notation1 SYSTEM "file.txt">
 <!NOTATION notation2 PUBLIC "some notation">
 <!ELEMENT doc (p*)>
 <!ENTITY example "<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;).</p>">
]>
<doc xml:lang="foo">&example;</doc>

```

## XML Parser for C++ Example 5: XML — NSEExample.xml

The following example file, `NSEExample.xml`, uses namespaces.

```

<!DOCTYPE doc [

```

```

<!ELEMENT doc (child*)>
<!ATTLIST doc xmlns:nsprefix CDATA #IMPLIED>
<!ATTLIST doc xmlns CDATA #IMPLIED>
<!ATTLIST doc nsprefix:al CDATA #IMPLIED>
<!ELEMENT child (#PCDATA)>
]>
<doc nsprefix:al = "v1" xmlns="http://www.w3c.org"
xmlns:nsprefix="http://www.oracle.com">
<child>
This element inherits the default Namespace of doc.
</child>
</doc>

```

## XML Parser for C++ Example 6: C++ — DOMSample.cpp

This example contains the C++ source code for DOMSample.cpp.

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
```

```

//
// NAME
// DOMSample.cpp
//
// DESCRIPTION
// Sample usage of C++ XML parser via DOM interface
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
// none
//

#include <iostream.h>
#include <string.h>

#ifdef ORAXMLDOM_ORACLE
include <oraxml.h>
#endif

#define DOCUMENT"cleo.xml"
#define DEFAULT_SPEAKER"Soothsayer"

void dump(Node *node);

```

```

void dumpspeech(Node *node);

char *speaker;
char *act, *scene;
uword n_speech;

int main(int argc, char **argv)
{
 XMLParser parser;
 ub4 flags;
 uword ecode;
 flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

 cout << "XML C++ DOM sample\n";

 speaker = (argc > 1) ? argv[1] : DEFAULT_SPEAKER;

 cout << "Initializing XML package...\n";

 if (ecode = parser.xmlinit())
 {
 cout << "Failed to initialize XML parser, error " << ecode;
 return 1;
 }

 cout << "Parsing '" << DOCUMENT << "'...\n";
 cout.flush();
 if (ecode = parser.xmlparse((oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;

 cout << "Dumping " << speaker << " speeches...\n";
 cout.flush();
 cout << "-----\n";
 act = scene = "";
 n_speech = 0;
 dump(parser.getDocumentElement());

 (void) parser.xmlterm();// terminate LPX package

 return 0;
}

void dump(Node *node)
{
 Node *title, *speak;

```

```

char *name, *who;
uword i, n_nodes;

name = (char *) node->getName();
if (!strcmp((char *) name, "ACT"))
{
title = node->getFirstChild();
act = (char *) title->getFirstChild()->getValue();
}
else if (!strcmp((char *) name, "SCENE"))
{
title = node->getFirstChild();
scene = (char *) title->getFirstChild()->getValue();
}
else if (!strcmp((char *) name, "SPEECH"))
{
speak = node->getFirstChild();
who = (char *) speak->getFirstChild()->getValue();
if (!strcmp(who, speaker))
dumpspeech(node);
}

if (node->hasChildNodes())
{
n_nodes = node->numChildNodes();
for (i = 0; i < n_nodes; i++)
dump(node->getChildNode(i));
}
}

// <SPEECH>
// <SPEAKER>Soothsayer</SPEAKER>
// <LINE>Your will?</LINE>
// </SPEECH>

// <SPEECH>
// <SPEAKER>CLEOPATRA</SPEAKER>
// <LINE><STAGEDIR>Aside to DOMITIUS ENOBARBUS</STAGEDIR> What means
this?</LINE>
// </SPEECH>

void dumpspeech(Node *node)
{
Node *kid, *part, *partkid;
uword i, j, n_node, n_part;

```

```
 oratext *partname, *partval;

 if (n_speech++)
 cout << "\n";
 cout << act << ", " << scene << "\n";
 n_node = node->numChildNodes();
 for (i = 0; i < n_node; i++)// skip speaker
 {
 kid = node->getChildNode(i);// line #i
 if (!strcmp((char *) kid->getName(), "LINE"))
 {
 n_part = kid->numChildNodes();
 for (j = 0; j < n_part; j++)
 {
 part = kid->getChildNode(j);
 if (part->getType() == TEXT_NODE)
 cout << " " << (char *) part->getValue() << "\n";
 else
 {
 partname = part->getName();
 partval = part->getFirstChild()->getValue();
 if (!strcmp((char *) partname, "STAGEDIR"))
 cout << " [" << (char *) partval << "]\n";
 else
 cout << " {" << (char *) partval << "}\n";
 }
 }
 }
 }
 cout.flush();
 }

// end of DOMSample.c
```

## XML Parser for C++ Example 7: C++ — DOMSample.std

DOMSample.std shows the expected output from DOMSample.cpp.

```
XML C++ DOM sample
Initializing XML package...
Parsing 'cleo.xml'...
Dumping Soothsayer speeches...
```

```

ACT I, SCENE II. The same. Another room.
Your will?
```

ACT I, SCENE II. The same. Another room.  
In nature's infinite book of secrecy  
A little I can read.

ACT I, SCENE II. The same. Another room.  
I make not, but foresee.

ACT I, SCENE II. The same. Another room.  
You shall be yet far fairer than you are.

ACT I, SCENE II. The same. Another room.  
You shall be more believing than beloved.

ACT I, SCENE II. The same. Another room.  
You shall outlive the lady whom you serve.

ACT I, SCENE II. The same. Another room.  
You have seen and proved a fairer former fortune  
Than that which is to approach.

ACT I, SCENE II. The same. Another room.  
If every of your wishes had a womb.  
And fertile every wish, a million.

ACT I, SCENE II. The same. Another room.  
Your fortunes are alike.

ACT I, SCENE II. The same. Another room.  
I have said.

ACT II, SCENE III. The same. OCTAVIUS CAESAR'S house.  
Would I had never come from thence, nor you Thither!

ACT II, SCENE III. The same. OCTAVIUS CAESAR'S house.  
I see it in  
My motion, have it not in my tongue: but yet  
Hie you to Egypt again.

ACT II, SCENE III. The same. OCTAVIUS CAESAR'S house.  
Caesar's.  
Therefore, O Antony, stay not by his side:  
Thy demon, that's thy spirit which keeps thee, is  
Noble, courageous high, unmatchable,  
Where Caesar's is not; but, near him, thy angel

Becomes a fear, as being o'erpower'd: therefore  
Make space enough between you.

ACT II, SCENE III. The same. OCTAVIUS CAESAR's house.  
To none but thee; no more, but when to thee.  
If thou dost play with him at any game,  
Thou art sure to lose; and, of that natural luck,  
He beats thee 'gainst the odds: thy lustre thickens,  
When he shines by: I say again, thy spirit  
Is all afraid to govern thee near him;  
But, he away, 'tis noble.

## XML Parser for C++ Example 8: C++ — SAXSample.cpp

This example contains the C++ source code for `SAXSample.cpp`.

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
//
// NAME
// SAXSample.cpp
//
// DESCRIPTION
// Sample usage of C++ XML parser via SAX interface
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
// none
//

#include <iostream.h>
#include <string.h>

#ifdef ORAXMLDOM_ORACLE
include <oraxml.dom.h>
#endif

#define DOCUMENT"cleo.xml"
#define MAX_STRING128
#define MAX_SPEAKER20

oratext elem[MAX_STRING], last_elem[MAX_STRING];
uword n_speaker;
```



```
oratext *speakers[MAX_SPEAKER];
size_t speakerlen[MAX_SPEAKER];

/* SAX callback functions */

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);

xmlsaxcb saxcb = {
 startDocument,
 endDocument,
 startElement,
 endElement,
 characters
};

int main()
{
 XMLParser parser;
 ub4 flags;
 uword ecode;
 flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

 cout << "XML C++ SAX sample\n";

 cout << "Initializing XML package...\n";

 if (ecode = parser.xmlinit((oratext *) 0,
// encoding
 (void (*)(void *, const oratext *, ub4)) 0,
 (void *) 0,
// msghdlr ctx
 (xmlsaxcb *) &saxcb))
// SAX callback
 {
 cout << "Failed to initialize XML parser, error " << ecode;
 return 1;
 }

 cout << "Parsing '" << DOCUMENT << "' and showing speakers by scene...\n";
 cout.flush();
}
```

```
 if (ecode = parser.xmlparse((oratext *) DOCUMENT, (oratext *) 0, flags))
return 1;

 (void) parser.xmlterm();
// terminate LPX package

 return 0;
}

sword startDocument(void *ctx)
{
 cout << "startDocument\n";
 return 0;
}

sword endDocument(void *ctx)
{
 cout << "endDocument\n";
 return 0;
}

sword startElement(void *ctx, const oratext *name,
const struct xmlnodes *attrs)
{
 strcpy((char *) last_elem, (char *) elem);
 strcpy((char *) elem, (char *) name);
 return 0;
}

sword endElement(void *ctx, const oratext *name)
{
 uword i;

 if (!strcmp((char *) name, "SCENE"))
 {
for (i = 0; i < n_speaker; i++)
 {
 cout << " ";
 cout.write(speakers[i], speakerlen[i]);
 cout << "\n";
 }
 }
 return 0;
}
}
```

```

sword characters(void *ctx, const oratext *ch, size_t len)
{
 uword i;

 if (!strcmp((char *) elem, "TITLE"))
 {
if (!strcmp((char *) last_elem, "ACT"))
 {
 cout << "\n--- ";
 cout.write(ch, len);
 cout << " ---\n\n";
 }
else if (!strcmp((char *) last_elem, "SCENE"))
 {
 n_speaker = 0;
 cout << " ";
 cout.write(ch, len);
 cout << "\n";
 }
 }
 else if (!strcmp((char *) elem, "SPEAKER"))
 {
if (n_speaker < MAX_SPEAKER)
 {
 for (i = 0; i < n_speaker; i++)
if ((len == speakerlen[i] && !strcmp((char *) speakers[i],
(char *) ch, len))
 break;
 if (!n_speaker || (i == n_speaker))
 {
speakers[n_speaker] = (oratext *) ch;
speakerlen[n_speaker++] = len;
 }
 }
 }
 return 0;
}

// end of SAXSample.cc

```

## XML Parser for C++ Example 9: C++ — SAXSample.std

SAXSample.std shows the expected output from SAXSample.cpp.

XML C++ SAX sample

```
Initializing XML package...
Parsing 'cleo.xml' and showing speakers by scene...
startDocument

--- ACT I ---

SCENE I. Alexandria. A room in CLEOPATRA's palace.
 PHILO
 CLEOPATRA
 MARK ANTONY
 Attendant
 DEMETRIUS
SCENE II. The same. Another room.
 CHARMIAN
 ALEXAS
 Soothsayer
 DOMITIUS ENOBARBUS
 IRAS
 CLEOPATRA
 Messenger
 MARK ANTONY
 First Attendant
 Second Attendant
 Second Messenger
SCENE III. The same. Another room.
 CLEOPATRA
 CHARMIAN
 MARK ANTONY
SCENE IV. Rome. OCTAVIUS CAESAR's house.
 OCTAVIUS CAESAR
 LEPIDUS
 Messenger
SCENE V. Alexandria. CLEOPATRA's palace.
 CLEOPATRA
 CHARMIAN
 MARDIAN
 ALEXAS

--- ACT II ---

...
...
--- ACT V ---
SCENE I. Alexandria. OCTAVIUS CAESAR's camp.
 OCTAVIUS CAESAR
```

```

DOLABELLA
DERCETAS
AGRIPPA
MECAENAS
Egyptian
PROCULEIUS
All
SCENE II. Alexandria. A room in the monument.
CLEOPATRA
PROCULEIUS
GALLUS
IRAS
CHARMIAN
DOLABELLA
OCTAVIUS CAESAR
SELEUCUS
Guard
Clown
First Guard
Second Guard
endDocument

```

## XML Parser for C++ Example 10: C++ — DOMNamespace.cpp

This example contains the C++ source code for DOMNamespace.cpp.

```

// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
///
// NAME
// DOMNamespace.cpp
//
// DESCRIPTION
// This file demonstates a simple use of the parser and Namespace
// extensions to the DOM APIs.
//
// The XML file that is given to the application is parsed and the
// elements and attributes in the document are printed.
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
// none
///

```

```
#include <iostream.h>

#ifdef ORAXMLDOM_ORACLE
include <oraxmlDOM.h>
#endif

#define DOCUMENT "NSEExample.xml"

void dump(Node *node);
void dumpattrs(Node *node);

//
// main
//

int main()
{
 XMLParser parser;
 ub4 flags;
 uword ecode;
 flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

 cout << "\nXML C++ DOM Namespace\n";
 cout << "Initializing XML package...\n";
 if (ecode = parser.xmlinit())
 {
 cout << "Failed to initialize XML parser, error " << ecode;
 return 1;
 }

 cout << "Parsing '" << DOCUMENT << "'...\n";
 cout.flush();
 if (ecode = parser.xmlparse((oraxml *) DOCUMENT, (oraxml *) 0, flags))
 return 1;

 cout << "\nThe elements are:\n";
 dump(parser.getDocumentElement());
 (void) parser.xmlterm();// terminate LPX package
 return 0;
}

//
// dump
//
```

```

void dump(Node *node)
{
 uword i, n_nodes;
 NodeList *nodes;
 size_t nn;

 String qName;
 String localName;
 String nsName;
 String prefix;

 if (node == NULL)
 return;
 if (nodes = node->getChildNodes())
 {
 for (nn = node->numChildNodes(), i=0; i < nn; i++)
 {
 // Use the methods getQualifiedName(), getLocalName(),
 // getPrefix(), and getNamespace() to get Namespace
 // information.

 qName = prefix = localName = nsName = (oratext *)" ";

 if (node->getQualifiedName() != (oratext *)NULL)
 qName = node->getQualifiedName();

 if (node->getPrefix() != (oratext *)NULL)
 prefix = node->getPrefix();

 if (node->getLocal() != (oratext *)NULL)
 localName = node->getLocal();

 if (node->getNamespace() != (oratext *)NULL)
 nsName = node->getNamespace();
 cout << " ELEMENT Qualified Name: " << (char *)qName << "\n";
 cout << " ELEMENT Prefix : " << (char *)prefix << "\n";
 cout << " ELEMENT Local Name : " << (char *)localName << "\n";
 cout << " ELEMENT Namespace : " << (char *)nsName << "\n";
 dumpattrs(node);
 dump(node->getChildNode(i));
 }
 }
}

```

```
//
// dumpattrs
//

void dumpattrs(Node *node)
{
 NamedNodeMap *attrs;
 Attr *a;
 uword i;
 size_t na;

 oratext *qname;
 oratext *namespace;
 oratext *local;
 oratext *prefix;
 oratext *value;
 if (attrs = node->getAttributes())
 {
 cout << "\n ATTRIBUTES: \n";
 for (na = attrs->getLength(), i = 0; i < na; i++)
 {
 /* get attr qualified name, local name, namespace, and prefix */

 a = (Attr *)attrs->item(i);
 qname = namespace = local = prefix = value = (oratext*)" ";
 if (a->getQualifiedName() != (oratext*)NULL)
 qname = a->getQualifiedName();
 if (a->getNamespace() != (oratext*)NULL)
 namespace = a->getNamespace();
 if (a->getLocal() != (oratext*)NULL)
 local = a->getLocal();
 if (a->getPrefix() != (oratext*)NULL)
 prefix = a->getPrefix();
 if (a->getValue() != (oratext*)NULL)
 value = a->getValue();

 cout << " " << (char*)qname << " = " << (char*)value << "\n";
 cout << " Namespace : " << (char*)namespace << "\n";
 cout << " Local Name: " << (char*)local << "\n";
 cout << " Prefix : " << (char*)prefix << "\n\n";
 }
 }
 cout << "\n";
}
```



## XML Parser for C++ Example 11: C++ — DOMNamespace.std

DOMNamespace.std shows the expected output from DOMNamespace.cpp.

```
XML C++ DOM Namespace
Initializing XML package...
Parsing 'NSExample.xml'...
```

The elements are:

```
ELEMENT Qualified Name: doc
ELEMENT Prefix :
ELEMENT Local Name : doc
ELEMENT Namespace : http://www.w3c.org
```

ATTRIBUTES:

```
 namespace:al = v1
 Namespace : http://www.oracle.com
 Local Name: al
 Prefix : namespace
```

```
 xmlns = http://www.w3c.org
 Namespace :
 Local Name: xmlns
 Prefix :
```

```
 xmlns:namespace = http://www.oracle.com
 Namespace :
 Local Name: namespace
 Prefix : xmlns
```

```
ELEMENT Qualified Name: child
ELEMENT Prefix :
ELEMENT Local Name : child
ELEMENT Namespace : http://www.w3c.org
```

## XML Parser for C++ Example 12: C++ — SAXNamespace.cpp

This example contains the C++ source code for the SAXNamespace program.

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
//
// NAME
// DOMNamespace.cpp
//
// DESCRIPTION
// This file demonstrates a simple use of the parser and Namespace
```

```

// extensions to the SAX APIs.
// The XML file that is given to the application is parsed and the
// elements and attributes in the document are printed.
//
// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
// none
////////////////////////////////////

#include <iostream.h>

#ifndef ORAXMLDOM_ORACLE
include <oraxmlDOM.h>
#endif

#define DOCUMENT "NSEExample.xml"

/*-----
 FUNCTION PROTOTYPES
-----*/

int startDocument(void *ctx);
int endDocument(void *ctx);
int endElement(void *ctx, const oratext *name);
int nsStartElement(void *ctx, const oratext *qname,
 const oratext *local,
 const oratext *nsp,
 const struct xmlnodes *attrs);

/* SAX callback structure */

xmlsaxcb saxcb = {
 startDocument,
 endDocument,
 0,
 endElement,
 0,
 0,
 0,
 0,
 nsStartElement,
 0, 0, 0, 0, 0, 0, 0, 0
}

```

```

};

/* SAX callback context */
/*
typedef struct {
 xmlctx *ctx;
 uword depth;
} cbctx;
*/

/*-----
 MAIN
-----*/

int main()
{
 XMLParser parser;
 ub4 flags;
 uword ecode;
 flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;
 cout << "XML C++ SAX Namespace\n";
 cout << "Initializing XML package...\n";
 if (ecode = parser.xmlinit((oralex * *) 0, // encoding
 (void (*)(void *, const oralex *, ub4)) 0,
 (void *) 0, // msghdlr ctx
 (xmlsaxcb *) &saxcb)) // SAX callback
 {
 cout << "Failed to initialize XML parser, error " << ecode;
 return 1;
 }

 /* parse the document */

 cout << "Parsing '" << DOCUMENT << "'...\n";
 cout.flush();
 if (ecode = parser.xmlparse((oralex *) DOCUMENT, (oralex *) 0, flags))
 return 1;

 (void) parser.xmlterm();// terminate LPX package

 return 0;
}

/*-----
 SAX Interface
-----*/

```

```

int startDocument(void *ctx)
{
 cout << "\nStartDocument\n\n";
 return 0;
}

int endDocument(void *ctx)
{
 cout << "\nEndDocument\n";
 return 0;
}

int endElement(void *ctx, const oratext *name)
{
 cout << "\nELEMENT Name : " << (char*)name << "\n";
 return 0;
}

int nsStartElement(void *ctx, const oratext *qname, const oratext *local,
 const oratext *nsp, const struct xmlnodes *attrs)
{
 xmlnode *attr;
 uword i;
 oratext *aqname;
 oratext *alocal;
 oratext *anamespace;
 oratext *aprefix;
 oratext *avalue;

 /*
 * Use the functions getXXXQualifiedName(), getXXXLocalName(), and
 * getXXXNamespace() to get Namespace information.
 */

 if (qname == (oratext*)NULL)
 qname = (oratext*)" ";
 if (local == (oratext*)NULL)
 local = (oratext*)" ";
 if (nsp == (oratext*)NULL)
 nsp = (oratext*)" ";

 cout << "ELEMENT Qualified Name: " << (char*)qname << "\n";
 cout << "ELEMENT Local Name : " << (char*)local << "\n";
 cout << "ELEMENT Namespace : " << (char*)nsp << "\n";
}

```

```

 if (attrs)
 {
 for (i = 0; i < numAttributes(attrs); i++)
 {
 attr = getAttributeIndex(attrs,i);
 aqname = alocal = anamespace = aprefix = avalue = (oratest *) " ";

 if (getAttrQualifiedName(attr))
 aqname = (oratest *) getAttrQualifiedName(attr);
 if (getAttrPrefix(attr))
 aprefix = (oratest *) getAttrPrefix(attr);
 if (getAttrLocal(attr))
 alocal = (oratest *) getAttrLocal(attr);
 if (getAttrNamespace(attr))
 anamespace = (oratest *) getAttrNamespace(attr);
 if (getAttrValue(attr))
 avalue = (oratest *) getAttrValue(attr);

 cout << " ATTRIBUTE Qualified Name : " << (char*)aqname << "\n";
 cout << " ATTRIBUTE Prefix : " << (char*)aprefix << "\n";
 cout << " ATTRIBUTE Local Name : " << (char*)alocal << "\n";
 cout << " ATTRIBUTE Namespace : " << (char*)anamespace << "\n";
 cout << " ATTRIBUTE Value : " << (char*)avalue << "\n";
 cout << "\n";
 }
 }
 return 0;
}

```

## XML Parser for C++ Example 13: C++ — SAXNamespace.std

SAXNamespace.std shows the expected output from SAXNamespace.cpp.

```

XML C++ SAX Namespace
Initializing XML package...
Parsing 'NSExample.xml'...

StartDocument

ELEMENT Qualified Name: doc
ELEMENT Local Name : doc
ELEMENT Namespace : http://www.w3c.org
ATTRIBUTE Qualified Name : nsprefix:a1
ATTRIBUTE Prefix : nsprefix
ATTRIBUTE Local Name : a1

```

```
ATTRIBUTE Namespace : http://www.oracle.com
ATTRIBUTE Value : v1

ATTRIBUTE Qualified Name : xmlns
ATTRIBUTE Prefix :
ATTRIBUTE Local Name : xmlns
ATTRIBUTE Namespace :
ATTRIBUTE Value : http://www.w3c.org

ATTRIBUTE Qualified Name : xmlns:nsprefix
ATTRIBUTE Prefix : xmlns
ATTRIBUTE Local Name : nsprefix
ATTRIBUTE Namespace :
ATTRIBUTE Value : http://www.oracle.com

ELEMENT Qualified Name: child
ELEMENT Local Name : child
ELEMENT Namespace : http://www.w3c.org

ELEMENT Name : child
ELEMENT Name : doc
EndDocument
```

## XML Parser for C++ Example 14: C++ — FullDOM.cpp

This example contains the C++ source code for FullDOM.cpp.

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.
//
// NAME
// FullDOM.cpp
//
// DESCRIPTION
// Sample code to test full C++ DOM interface
//

#include <iostream.h>

#ifndef ORAXMLDOM_ORACLE
include <oraxmlDOM.h>
#endif

#define TEST_DOC(oratext *) "FullDOM.xml"
```

```
void dump(Node *node, uword level);
void dumpnode(Node *node, uword level);

static char *ntypename[] = {
 "0",
 "ELEMENT",
 "ATTRIBUTE",
 "TEXT",
 "CDATA",
 "ENTREF",
 "ENTITY",
 "PI",
 "COMMENT",
 "DOCUMENT",
 "DTD",
 "DOCFRAG",
 "NOTATION"
};

#define FAIL { cout << "Failed!\n"; return 1; }

int main()
{
 XMLParser parser;
 Document *doc;
 Element *root, *elem, *subelem;
 Attr *attr, *attr1, *attr2, *gleep1, *gleep2;
 Text *text, *subtext;
 Node *node, *pi, *comment, *entref, *cdata, *clone,
 *deep_clone, *frag, *fragelem, *fragtext, *sub2,
 *fish, *food, *food2, *repl;
 NodeList *subs, *nodes;
 NamedNodeMap *attrs, *notes, *entities;
 DocumentType *dtd;
 uword i, ecode, level;

 cout << "XML C++ Full DOM test\n";
 cout << "Initializing XML parser...\n";

 if (ecode = parser.xmlinit())
 {
 cout << "Failed to initialize XML parser, error " << ecode << "\n";
 return 1;
 }
}
```

```

 cout << "\nCreating new document...\n";
 if (!(doc = parser.createDocument()))
FAIL

 cout << "Document from root node:\n";
 dump(parser.getDocument(), 0);

 cout << "\nCreating root element ('ROOT')...\n";
 if (!(elem = doc->createElement((oratext *) "ROOT")))
FAIL

 cout << "Setting as root element...\n";
 if (!doc->appendChild(elem))
FAIL

 cout << "Document from 'ROOT' element:\n";
 dump(root = parser.getDocumentElement(), 0);
 cout << "Adding 7 children to 'ROOT' element...\n";
 if (!(text = doc->createTextNode((oratext *) "Gibberish")) ||
 !elem->appendChild(text))
FAIL
 if (!(comment = doc->createComment((oratext*) "Bit warm today, innit?")) ||
 !elem->appendChild(comment))
FAIL

 if (!(pi = doc->createProcessingInstruction((oratext *) "target",
(oratext *) "PI-contents")) ||
 !elem->appendChild(pi))
FAIL

 if (!(cdata = doc->createCDATASection((oratext *) "See DATA")) ||
 !elem->appendChild(cdata))
FAIL

 if (!(entref = doc->createEntityReference((oratext *) "EntRef")) ||
 !elem->appendChild(entref))
FAIL

 if (!(fish = doc->createElement((oratext *) "FISH")) ||
 !elem->appendChild(fish))
FAIL

 if (!(food = doc->createElement((oratext *) "FOOD")) ||
 !elem->appendChild(food))
FAIL

```



```

cout << "Document from 'ROOT' element with its 7 children:\n";
dump(root, 0);

cout << "\nTesting node insertion...\n";
cout << "Adding 'Pre-Gibberish' text node and 'Ask about the weather'
comment node...\n";
if (!(node = doc->createTextNode((oratext *) "Pre-Gibberish")) ||
 !elem->insertBefore(node, text))
FAIL

if (!(node = doc->createComment((oratext *) "Ask about the weather:")) ||
 !elem->insertBefore(node, comment))
FAIL

cout << "Document from 'ROOT' element:\n";
dump(root, 0);
cout << "Document from 'ROOT' element:\n";
dump(root, 0);
cout << "Document from 'ROOT' element:\n";
dump(root, 0);
cout << "\nTesting nextSibling links starting at first child...\n";
for (node = elem->getFirstChild();
 node;
 node = node->getNextSibling())dump(node, 1);
cout << "\nTesting previousSibling links starting at last child...\n";
for (node = elem->getLastChild();
 node;
 node = node->getPreviousSibling())dump(node, 1);

cout << "\nTesting setting node value...\n";
cout << "Original node:\n";
dump(pi, 1);
pi->setValue((oratext *) "New PI contents");
cout << "Node after new value:\n";
dump(pi, 1);

cout << "\nAdding another element level, i.e., 'SUB'...\n";
if (!(subelem = doc->createElement((oratext *) "SUB")) ||
 !elem->insertBefore(subelem, cdata) ||
 !(subtext = doc->createTextNode((oratext *) "Lengthy SubText")) ||
 !subelem->appendChild(subtext))
FAIL

cout << "Document from 'ROOT' element:\n";
dump(root, 0);

```

```

 cout << "\nAdding a second 'SUB' element...\n";
 if (!(sub2 = doc->createElement((oratext *) "SUB")) ||
!elem->insertBefore(sub2, cdata))
FAIL

 cout << "Document from 'ROOT' element:\n";
 dump(root, 0);

 cout << "\nGetting all SUB nodes - note the distinct hex addresses...\n";
 if (!(subs = doc->getElementsByTagName(root, (oratext *) "SUB")))
FAIL
 for (i = 0; i < subs->getLength(); i++)
dumpnode(subs->item(i), 1);

 cout << "\nTesting parent links...\n";
 for (level = 1, node = subtext; node; node = node->getParentNode(), level++)
dumpnode(node, level);

 cout << "\nTesting owner document of node...\n";
 dumpnode(subtext, 1);
 dumpnode(subtext->getOwnerDocument(), 1);

 cout << "\nTesting node replacement...\n";
 if (!(node = doc->createTextNode((oratext *) "REPLACEMENT, 1/2 PRICE")) ||
!pi->replaceChild(node))
FAIL

 cout << "Document from 'ROOT' element:\n";
 dump(root, 0);

 cout << "\nTesting node removal...\n";
 if (!entref->removeChild())
FAIL

 cout << "Document from 'ROOT' element:\n";
 dump(root, 0);
 cout << "\nNormalizing...\n";
 elem->normalize();
 cout << "Document from 'ROOT' element:\n";
 dump(root, 0);
 cout << "\nCreating and populating document fragment...\n";
 if (!(frag = doc->createDocumentFragment()) ||
!(fragelem = doc->createElement((oratext *) "FragElem")) ||
!(fragtext = doc->createTextNode((oratext *) "FragText")) ||

```

```

!frag->appendChild(fragelem) ||
!frag->appendChild(fragtext))
FAIL
 dump(frag, 1);
 cout << "Insert document fragment...\n";
 if (!elem->insertBefore(frag, comment))
FAIL
 dump(elem, 1);

 cout << "\nCreate two attributes...\n";
 if (!(attr1 = doc->createAttribute((oratest*)"Attr1", (oratest*)"Value1")) ||
!(attr2 = doc->createAttribute((oratest*)"Attr2", (oratest*)"Value2")))
FAIL
 cout << "Setting attributes...\n";
 if (!subelem->setAttributeNode(attr1, NULL) ||
!subelem->setAttributeNode(attr2, NULL))
FAIL
 dump(subelem, 1);

 cout << "\nAltering attribute1 value...\n";
 attr1->setValue((oratest *) "New1");
 dump(subelem, 1);

 cout << "\nFetching attribute by name (Attr2)...\n";
 if (!(node = subelem->getAttributeNode((oratest *) "Attr2")))
FAIL
 dump(node, 1);

 cout << "\nRemoving attribute by name (Attr1)...\n";
 subelem->removeAttribute((oratest *) "Attr1");
 dump(subelem, 1);

 cout << "\nAdding new attribute...\n";
 if (!subelem->setAttribute((oratest *) "Attr3", (oratest *) "Value3"))
FAIL
 dump(subelem, 1);

 cout << "\nRemoving attribute by pointer (Attr2)...\n";
 if (!subelem->removeAttributeNode(attr2))
FAIL
 dump(subelem, 1);

 cout << "\nAdding new attribute w/same name (test replacement)...\n";
 dump(subelem, 1);
 if (!(attr = doc->createAttribute((oratest*)"Attr3", (oratest*)"Zoo3")))

```

```

FAIL
 if (!subelem->setAttributeNode(attr, NULL))
FAIL
 dump(subelem, 1);

 cout << "\nTesting node (attr) set by name...\n";
 cout << "Adding 'GLEEP' attr and printing out hex addresses of node set\n";
 attrs = subelem->getAttributes();
 if (!(gleep1=doc->createAttribute((oratext*)"GLEEP", (oratext*)"GLEEP1")) ||
!attrs->setNamedItem(gleep1, NULL))
FAIL
 dump(subelem, 0);

 cout << "\nTesting node (attr) set by name...\n";
 cout << "Replacing 'GLEEP' element - note the changed hex address\n";
 if (!(gleep2=doc->createAttribute((oratext*)"GLEEP", (oratext*)"GLEEP2")) ||
!attrs->setNamedItem(gleep2, &repl))
FAIL
 dump(subelem, 0);
 cout << "Replaced node was:\n";
 dump(repl, 1);

 cout << "\nTesting node removal by name...\n";
 cout << "Removing 'GLEEP' attribute\n";
 if (!attrs->removeNamedItem((oratext *) "GLEEP"))
FAIL
 dump(subelem, 0);

 cout << "\nOriginal SubROOT...\n";
 dump(subelem, 1);
 cout << "Cloned SubROOT (not deep)...\n";
 clone = subelem->cloneNode(FALSE);
 dump(clone, 1);
 cout << "Cloned SubROOT (deep)...\n";
 deep_clone = subelem->cloneNode(TRUE);
 dump(deep_clone, 1);

 cout << "\nSplitting text...\n";
 dump(subelem, 1);
 subtext->splitText(3);
 dump(subelem, 1);

 cout << "\nTesting string operations...\n";
 cout << " CharData = \"\" << (char *) subtext->getData() << "\"\n";
 cout << "Setting new data...\n";

```

```

subtext->setData((oratest *) "0123456789");
cout << " CharData = \"\" << (char *) subtext->getData() << "\\n";
cout << " CharLength = \"\" << (int) subtext->getLength() << "\\n";
cout << " Substring(0,5) = \"\" <<
(char *) subtext->substringData(0, 5) << "\\n";
 cout << " Substring(8,2) = \"\" <<
(char *) subtext->substringData(8, 2) << "\\n";
 cout << "Appending data...\n";
subtext->appendData((oratest *) "ABCDEF");
cout << " CharData = \"\" << (char *) subtext->getData() << "\\n";
cout << "Inserting data...\n";
subtext->insertData(10, (oratest *) "**foo*");
cout << " CharData = \"\" << (char *) subtext->getData() << "\\n";
cout << "Deleting data...\n";
subtext->deleteData(0, 10);
cout << " CharData = \"\" << (char *) subtext->getData() << "\\n";
cout << "Replacing data...\n";
subtext->replaceData(1, 3, (oratest *) "bamboozle");
cout << " CharData = \"\" << (char *) subtext->getData() << "\\n";

cout << "Cleaning up...\n";
parser.xmlclean();

if (parser.getDocument())
{
cout << "Problem, document is not gone!!\n";
return 1;
}

cout << "Parsing test document...\n";
if (ecode = parser.xmlparse(TEST_DOC, (oratest *) 0, 0))
{
cout << "Parse failed, code \" << ecode << "\\n\";
return ecode;
}

cout << "Document from root node:\n\" << flush;
dump(parser.getDocument(), 0);

cout << "Testing getNotations...\n\" << flush;
dtd = parser.getDocType();
if (notes = dtd->getNotations())
{
cout << "# of notations = \" << notes->getLength() << "\\n\" << flush;
for (i = 0; i < notes->getLength(); i++)

```

```

 dump(notes->item(i), 1);
 }
 else
cout << "No defined notations\n" << flush;

 cout << "Testing getEntities...\n" << flush;
 if (entities = dtd->getEntities())
 {
cout << "# of entities = " << entities->getLength() << "\n" << flush;
for (i = 0; i < entities->getLength(); i++)
 dump(entities->item(i), 1);
 }
 else
cout << "No defined entities\n" << flush;

 cout << "Cleaning up...\n";
 parser.xmlclean();

 if (parser.getDocument())
 {
cout << "Problem, document is not gone!!\n";
return 1;
 }

 cout << "\nTerminating parser...\n";
 parser.xmlterm();

 cout << "Success.\n";
 return 0;
 }

void dump(Node *node, uword level)
{
 NodeList *nodes;
 uword i, n_nodes;

 if (node)
 {
dumpnode(node, level);
if (node->hasChildNodes())
 {
 nodes = node->getChildNodes();
 n_nodes = node->numChildNodes();
 for (i = 0; i < n_nodes; i++)
 dump(nodes->item(i), level + 1);
 }
}
}

```

```

 }
 }
}

void dumpnode(Node *node, uword level)
{
 const oratext *name, *value;
 short type;
 NamedNodeMap *attrs;
 Attr *attr;
 uword i, n_attrs;

 if (node)
 {
 for (i = 0; i <= level; i++)
 cout << " ";
 type = node->getType();
 cout << (char *) ntypename[type];
 if ((name = node->getName()) && (*name != '#'))
 cout << " \"" << (char *) name << "\"";
 if (value = node->getValue())
 cout << " = \"" << (char *) value << "\"";
 if ((type == ELEMENT_NODE) && (attrs = node->getAttributes()))
 {
 cout << " [";
 n_attrs = attrs->getLength();
 for (i = 0; i < n_attrs; i++)
 {
 if (i) cout << ", ";
 attr = (Attr *) attrs->item(i);
 cout << (char *) attr->getName();
 if (attr->getSpecified())
 cout << " ";
 cout << "=" << (char *) attr->getValue() << "\"";
 }
 cout << "];";
 }
 cout << "\n";
 }
}

// end of FullDOM.cpp

```

## XML Parser for C++ Example 15: C++ — FullDOM.std

The FullDOM.std example file shows the expected output from FullDOM.cpp

```
XML C++ Full DOM test
Initializing XML parser...

Creating new document...
Document from root node:
 DOCUMENT

Creating root element ('ROOT')...
Setting as root element...
Document from 'ROOT' element:
 ELEMENT "ROOT"
Adding 7 children to 'ROOT' element...
Document from 'ROOT' element with its 7 children:
 ELEMENT "ROOT"
 TEXT = "Gibberish"
 COMMENT = "Bit warm today, innit?"
 PI "target" = "PI-contents"
 CDATA = "See DATA"
 ENTREF "EntRef"
 ELEMENT "FISH"
 ELEMENT "FOOD"

Testing node insertion...
Adding 'Pre-Gibberish' text node and 'Ask about the weather' comment node...
Document from 'ROOT' element:
 ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 PI "target" = "PI-contents"
 CDATA = "See DATA"
 ENTREF "EntRef"
 ELEMENT "FISH"
 ELEMENT "FOOD"
Document from 'ROOT' element:
 ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 PI "target" = "PI-contents"
```



```
CDATA = "See DATA"
ENTREF "EntRef"
ELEMENT "FISH"
ELEMENT "FOOD"
Document from 'ROOT' element:
ELEMENT "ROOT"
TEXT = "Pre-Gibberish"
TEXT = "Gibberish"
COMMENT = "Ask about the weather:"
COMMENT = "Bit warm today, innit?"
PI "target" = "PI-contents"
CDATA = "See DATA"
ENTREF "EntRef"
ELEMENT "FISH"
ELEMENT "FOOD"
```

Testing nextSibling links starting at first child...

```
TEXT = "Pre-Gibberish"
TEXT = "Gibberish"
COMMENT = "Ask about the weather:"
COMMENT = "Bit warm today, innit?"
PI "target" = "PI-contents"
CDATA = "See DATA"
ENTREF "EntRef"
ELEMENT "FISH"
ELEMENT "FOOD"
```

Testing previousSibling links starting at last child...

```
ELEMENT "FOOD"
ELEMENT "FISH"
ENTREF "EntRef"
CDATA = "See DATA"
PI "target" = "PI-contents"
COMMENT = "Bit warm today, innit?"
COMMENT = "Ask about the weather:"
TEXT = "Gibberish"
TEXT = "Pre-Gibberish"
```

Testing setting node value...

Original node:

```
PI "target" = "PI-contents"
```

Node after new value:

```
PI "target" = "New PI contents"
```

Adding another element level, i.e., 'SUB'...

Document from 'ROOT' element:

```
ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 PI "target" = "New PI contents"
 ELEMENT "SUB"
 TEXT = "Lengthy SubText"
 CDATA = "See DATA"
 ENTREF "EntRef"
 ELEMENT "FISH"
 ELEMENT "FOOD"
```

Adding a second 'SUB' element...

Document from 'ROOT' element:

```
ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 PI "target" = "New PI contents"
 ELEMENT "SUB"
 TEXT = "Lengthy SubText"
 ELEMENT "SUB"
 CDATA = "See DATA"
 ENTREF "EntRef"
 ELEMENT "FISH"
 ELEMENT "FOOD"
```

Getting all SUB nodes - note the distinct hex addresses...

```
ELEMENT "SUB"
ELEMENT "SUB"
```

Testing parent links...

```
TEXT = "Lengthy SubText"
 ELEMENT "SUB"
 ELEMENT "ROOT"
 DOCUMENT
```

Testing owner document of node...

```
TEXT = "Lengthy SubText"
DOCUMENT
```

Testing node replacement...

Document from 'ROOT' element:

```
ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 TEXT = "REPLACEMENT, 1/2 PRICE"
 ELEMENT "SUB"
 TEXT = "Lengthy SubText"
 ELEMENT "SUB"
 CDATA = "See DATA"
 ENTREF "EntRef"
 ELEMENT "FISH"
 ELEMENT "FOOD"
```

Testing node removal...

Document from 'ROOT' element:

```
ELEMENT "ROOT"
 TEXT = "Pre-Gibberish"
 TEXT = "Gibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 TEXT = "REPLACEMENT, 1/2 PRICE"
 ELEMENT "SUB"
 TEXT = "Lengthy SubText"
 ELEMENT "SUB"
 CDATA = "See DATA"
 ELEMENT "FISH"
 ELEMENT "FOOD"
```

Normalizing...

Document from 'ROOT' element:

```
ELEMENT "ROOT"
 TEXT = "Pre-GibberishGibberish"
 COMMENT = "Ask about the weather:"
 COMMENT = "Bit warm today, innit?"
 TEXT = "REPLACEMENT, 1/2 PRICE"
 ELEMENT "SUB"
 TEXT = "Lengthy SubText"
 ELEMENT "SUB"
 CDATA = "See DATA"
 ELEMENT "FISH"
 ELEMENT "FOOD"
```

Creating and populating document fragment...

```
DOCFRAG
 ELEMENT "FragElem"
 TEXT = "FragText"
Insert document fragment...
 ELEMENT "ROOT"
 TEXT = "Pre-GibberishGibberish"
 COMMENT = "Ask about the weather:"
 ELEMENT "FragElem"
 TEXT = "FragText"
 COMMENT = "Bit warm today, innit?"
 TEXT = "REPLACEMENT, 1/2 PRICE"
 ELEMENT "SUB"
 TEXT = "Lengthy SubText"
 ELEMENT "SUB"
 CDATA = "See DATA"
 ELEMENT "FISH"
 ELEMENT "FOOD"

Create two attributes...
Setting attributes...
 ELEMENT "SUB" [Attr1*="Value1", Attr2*="Value2"]
 TEXT = "Lengthy SubText"

Altering attributel value...
 ELEMENT "SUB" [Attr1*="New1", Attr2*="Value2"]
 TEXT = "Lengthy SubText"

Fetching attribute by name (Attr2)...
 ATTRIBUTE "Attr2" = "Value2"

Removing attribute by name (Attr1)...
 ELEMENT "SUB" [Attr2*="Value2"]
 TEXT = "Lengthy SubText"

Adding new attribute...
 ELEMENT "SUB" [Attr2*="Value2", Attr3*="Value3"]
 TEXT = "Lengthy SubText"

Removing attribute by pointer (Attr2)...
 ELEMENT "SUB" [Attr3*="Value3"]
 TEXT = "Lengthy SubText"

Adding new attribute w/same name (test replacement)...
 ELEMENT "SUB" [Attr3*="Value3"]
 TEXT = "Lengthy SubText"
```

```

ELEMENT "SUB" [Attr3*="Zoo3"]
 TEXT = "Lengthy SubText"

```

Testing node (attr) set by name...

Adding 'GLEEP' attr and printing out hex addresses of node set

```

ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="GLEEP1"]
 TEXT = "Lengthy SubText"

```

Testing node (attr) set by name...

Replacing 'GLEEP' element - note the changed hex address

```

ELEMENT "SUB" [Attr3*="Zoo3", GLEEP*="GLEEP2"]
 TEXT = "Lengthy SubText"

```

Replaced node was:

```

ATTRIBUTE "GLEEP" = "GLEEP1"

```

Testing node removal by name...

Removing 'GLEEP' attribute

```

ELEMENT "SUB" [Attr3*="Zoo3"]
 TEXT = "Lengthy SubText"

```

Original SubROOT...

```

ELEMENT "SUB" [Attr3*="Zoo3"]
 TEXT = "Lengthy SubText"

```

Cloned SubROOT (not deep)...

```

ELEMENT "SUB" [Attr3*="Zoo3"]
 TEXT = "Lengthy SubText"

```

Cloned SubROOT (deep)...

```

ELEMENT "SUB" [Attr3*="Zoo3"]
 TEXT = "Lengthy SubText"

```

Splitting text...

```

ELEMENT "SUB" [Attr3*="Zoo3"]
 TEXT = "Lengthy SubText"
ELEMENT "SUB" [Attr3*="Zoo3"]
 TEXT = "Leng"
 TEXT = "thy SubText"

```

Testing string operations...

```

CharData = "Leng"

```

Setting new data...

```

CharData = "0123456789"
CharLength = 10
Substring(0,5) = "01234"
Substring(8,2) = "89"

```

Appending data...

```

CharData = "0123456789ABCDEF"
Inserting data...
CharData = "0123456789*foo*ABCDEF"
Deleting data...
CharData = "*foo*ABCDEF"
Replacing data...
CharData = "*bamboozle*ABCDEF"
Cleaning up...
Parsing test document...
Document from root node:
DOCUMENT
 DTD "doc"
 ELEMENT "doc" [xml:lang*="foo"]
 ELEMENT "p" [xml:space="preserve"]
 TEXT = "An ampersand (&) may be escaped
numerically (&) or with a general entity
(&)."
Testing getNotations...
of notations = 2
 NOTATION "notation1"
 NOTATION "notation2"
Testing getEntities...
of entities = 1
 ENTITY "example" = "<p>An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;).</p>"
Cleaning up...

Terminating parser...
Success.

```

## XML Parser for C++ Example 16: C++ — XSLSample.cpp

This example contains the C++ source code for XSLSample.cpp

// Copyright (c) Oracle Corporation 1999. All Rights Reserved.

```

//
// NAME
// XSLSample.cpp
//
// DESCRIPTION
// Sample usage of C++ XSL processor
//

```

```

// PUBLIC FUNCTION(S)
//
// PRIVATE FUNCTION(S)
//
// NOTES
// none
///

#ifndef ORAXMLDOM_ORACLE
include <oraxmlDOM.h>
#endif

int main(int argc, char **argv)
{
 XMLParser xmlpar, xslpar, respar;
 XSLProcessor xslproc;
 Node *result;
 ub4 flags;
 uword ecode;
 flags = XML_FLAG_VALIDATE | XML_FLAG_DISCARD_WHITESPACE;

 cout << "XSL processor sample\n";

 if (argc < 3)
 {
 cout << "Usage is XSLSample <xmlfile> <stylesheet>\n";
 return 1;
 }

 // Parse the XML file
 cout << "Parsing XML file " << argv[1] << "\n";
 if (ecode = xmlpar.xmlinit())
 {
 cout << "Failed to initialize XML parser, error " << ecode << "\n";
 return 1;
 }
 if (ecode = xmlpar.xmlparse((oraxmlDOM *) argv[1], (oraxmlDOM *) 0, flags))
 return 1;

 // Parse the Stylesheet file
 cout << "Parsing Stylesheet " << argv[2] << "\n";
 if (ecode = xslpar.xmlinit())
 {
 cout << "Failed to initialize XML parser, error " << ecode << "\n";
 return 1;
 }
}

```

```
 }
 if (ecode = xslpar.xmlparse((oratext *) argv[2], (oratext *) 0, flags))
 return 1;

 // Initialize the result context
 cout << "Initializing the result context\n";
 if (ecode = respar.xmlinit())
 {
 cout << "Failed to initialize XML parser, error " << ecode << "\n";
 return 1;
 }

 // XSL Processing
 cout << "XSL Processing\n";
 if (ecode = xslproc.xslprocess(&xmlpar, &xslpar, &respar, &result))
 {
 cout << "Failed in XSL Processing, error " << ecode << "\n";
 return 1;
 }

 // print the resultant tree
 cout.flush();
 xslproc.printres(&respar, result);

 // Terminate the parsers
 (void) xmlpar.xmlterm();
 (void) xslpar.xmlterm();
 (void) respar.xmlterm();

 return 0;
}
```

## XML Parser for C++ Example 17: C++ — XSLSample.std

This example shows the typical result output from XSLSample.cpp

```
<xsl:param name="size"/>
<xsl:param name="data"/>
<xsl:choose><xsl:when test="number(number($size) <
string-length(string($data)))">
 <xsl:value-of select="substring(string($data), 1, number($size))"/>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="string($data)"/>
</xsl:otherwise>
</xsl:choose>
```



```
</xsl:otherwise></xsl:choose>
<xsl:if test="number(number($size) > string-length(string($data)))">
<xsl:call-template name="pad">
<xsl:with-param name="padsz" select="number($size) -
string-length(string($data))"/>
</xsl:call-template></xsl:if></xsl:template><xsl:template match="/">
<xsl:text>#13;#10;</xsl:text><xsl:apply-templates select="//ROWSE
T/ROW/CUSTOMER"/>
 <xsl:text>#13;#10;</xsl:text>
</xsl:template><xsl:template match="CUSTOMER">
<xsl:call-template name="truncateorpad">
<xsl:with-param name="size" select="31"/>
<xsl:with-param name="data" select="."/>
</xsl:call-template>
</xsl:template>
</xsl:stylesheet>
```



---

## Using XML Schema Processor for C++

This chapter contains the following sections:

- [Oracle XML Schema Processor for C++ Features](#)
- [Invoking XML Schema Processor for C++](#)
- [XML Schema Processor for C++ Usage Diagram](#)
- [Running the Provided XML Schema Sample Application](#)

## Oracle XML Schema Processor for C++ Features

The XML Schema Processor for C++ is a companion component to the XML Parser for C++ that allows support to simple and complex datatypes into XML applications with Oracle. The Schema Processor supports the XML Schema Working Draft, with the goal being that it be 100% fully conformant when XML Schema becomes a W3C Recommendation. This makes writing custom applications that process XML documents straightforward in the Oracle environment, and means that a standards-compliant XML Schema Processor is part of the Oracle platform on every operating system where Oracle is ported.

**See Also:** [Chapter 20, "Using XML Parser for Java"](#), for more information about XML Schema and why you would want to use XML Schema.

XML Schema Processor for C++ has the following features:

- Supports simple and complex types
- Built upon the XML Parser for C++ v2
- The Schema processor is based on the April 7, 2000 version of the XML Schema Working Draft (in three parts):

The XML Schema Processor for C++ class is `XMLSchema`. The version described here is Oracle XML Schema Processor 1.0.1.0.0 (C++). The Oracle XML Schema Processor is an early adopter release and is written in C with a C++ wrapper. It includes the production release of the XML Parser for C v2.

**See Also:** *Oracle9i XML Reference*

## Requirements

XML Schema Processor for C++ runs on the following operating systems:

- Linux
- Solaris
- HP-UX
- NT 4 / Service Pack 3 (and above)

## Online Documentation

Documentation for Oracle XML Schema Processor for C++ is located in the doc directory in your install area.

## Standards Conformance

The parser conforms to the following standards:

- W3C recommendation for Extensible Markup Language (XML) 1.0
- W3C recommendation for Document Object Model Level 1.0
- W3C proposed recommendation for Namespaces in XML
- Simple API for XML (SAX) 1.0
- W3C recommendation for XSL Transformations (XSLT)
- W3C recommendation for XML Path Language (XPath)

## Using the Supported Character Sets

The XML Parser for C++ currently supports the following encodings:

- BIG5
- EBCDIC-CP-BE
- EBCDIC-CP-CA
- EBCDIC-CP-CH
- EBCDIC-CP-DK
- EBCDIC-CP-ES
- EBCDIC-CP-FI
- EBCDIC-CP-FR
- EBCDIC-CP-GB
- EBCDIC-CP-HE
- EBCDIC-CP-IS
- EBCDIC-CP-IT
- EBCDIC-CP-NL
- EBCDIC-CP-NO

- EBCDIC-CP-ROECE
- EBCDIC-CP-SE
- EBCDIC-CP-US
- EBCDIC-CP-WT
- EBCDIC-CP-YU
- EUC-JP
- GB2312
- ISO-10646-UCS-2
- ISO-8859-1 through 9
- KOI8-RUTF-8
- SHIFT\_JIS
- US-ASCII
- UTF-16

**See Also:** Appendix A, Character Sets, of the *Oracle9i Globalization and National Language Support Guide*, where, in addition, any character set specified in can be used.

To use these encodings, you must have the following set:

- The ORACLE\_HOME environment variable must be set and pointing to the location of your Oracle installation.
- The environment variables, ORA\_NLS, ORA\_NLS32, and ORA\_NLS33, must be set to point to the location of the NLS data files.
  - On Unix systems, this is usually `$ORACLE_HOME/ocommon/nls/admin/data`.
  - On Windows NT, this is usually `$ORACLE_HOME/nlsrtl/admin/nlsdata`.

The default encoding is UTF-8. It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to twice as fast as with multibyte character sets, such as UTF-8.

## XML Schema Processor for C++: Provided Software

Table 27-1 lists the supplied files and directories with this release:

**Table 27-1 XML Schema Processor for C++: Supplied Files**

Directory and Files	Description
license.html	Licensing agreement
readme.html	This file
bin/	Schema processor executable, "schema"
doc/	API documentation
include/	header files
lib/	XML/XSL/Schema & support libraries
mesg/	Error message files
sample/	Example usage of the Schema processor

Table 27-2 lists the included libraries:

**Table 27-2 XML Schema Processor for C++: Supplied Libraries**

Included Library	Description
libxml8.a	XML Parser/XSL Processor
libcore8.a	CORE functions
libnls8.a	National Language Support

## Invoking XML Schema Processor for C++

The XML Schema Processor can be called as an executable by invoking **bin/schema** in the install area. This takes two arguments:

- XML instance document
- Optionally, a default schema to apply

The Schema processor can also be invoked by writing code using the supplied APIs. The code must be compiled using the headers in the `include/` subdirectory and linked against the libraries in the `lib/` subdirectory. See `Makefile` in the `sample/` subdirectory for details on how to build your program.

An error message file is provided in the `mesg/` subdirectory. Currently, the only message file is in English although message files for other languages may be supplied in future releases.

### Set Environment Variable `OR_XML_MESG` to Point to Absolute Path

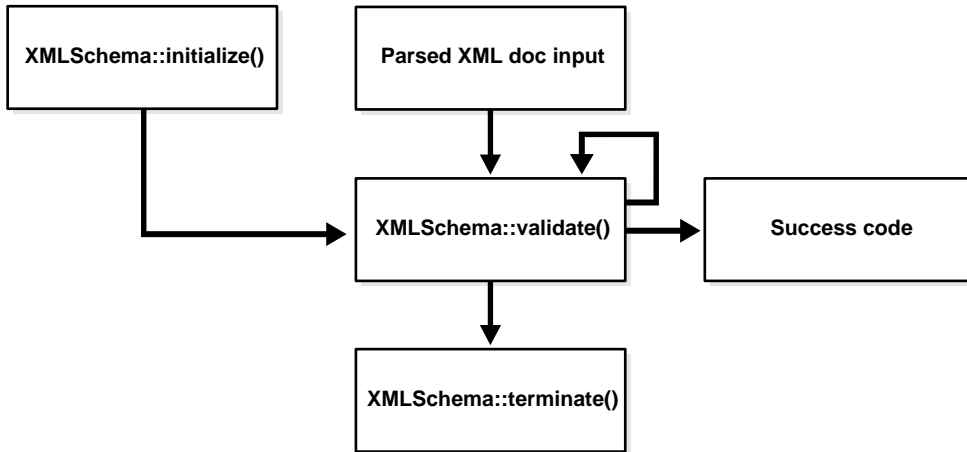
You should set the environment variable `ORA_XML_MESG` to point to the “**absolute**” path of the `mesg/` subdirectory. Alternately, if you have an `$ORACLE_HOME` installed, you may copy the contents of the `mesg/` subdirectory to the `$ORACLE_HOME/oracore/mesg` directory.

## XML Schema Processor for C++ Usage Diagram

[Figure 27-1](#) illustrates the calling sequence of XML Schema Processor for C++, as follows:

1. `XMLSchema.initialize()` method initializes the process.
2. The parsed XML document(s) inputs the Schema Processor.
3. `XMLSchema.validate()` validates the parsed XML document(s) until a success code results.
4. When validation completes, `XMLSchema.terminate()` method ends the process.



**Figure 27–1 XML Schema Processor for C++ Usage Diagram**

## Running the Provided XML Schema Sample Application

This directory contains a sample XML Schema application that illustrates how to use Oracle XML Schema Processor with its API. [Table 27–3](#) lists the provided sample files.

**Table 27–3 XML Schema for C++ Samples Provided**

Sample File	Description
<b>Makefile</b>	Makefile to build the sample programs and run them, verifying correct output.
<b>xsdtest.cpp</b>	Trivial program which invokes the XML Schema for C++ API
<b>car.{xsd,xml,std}</b>	Sample Schema, instance document, expected output respectively, after running xsdtest on them. See: <a href="#">"XML Schema for C++ Example 2: car.xsd"</a> on page 27-10 <a href="#">"XML Schema for C++ Example 3: car.xml"</a> on page 27-11 <a href="#">"XML Schema for C++ Example 4: car.std"</a> on page 27-11.

**Table 27-3 XML Schema for C++ Samples Provided**

Sample File	Description
<b>aq.{xsd,xml,std}</b>	<p>Second sample Schema's, instance document, expected output respectively, after running xsdtest on them. See:</p> <p>"XML Schema for C++ Example 5: aq.xsd" on page 27-12</p> <p>"XML Schema for C++ Example 6: aq.xml" on page 27-16</p> <p>"XML Schema for C++ Example 7: aq.std" on page 27-18</p>
<b>pub.{xsd,xml,std}</b>	<p>Third sample Schema's, instance document, expected output respectively, after running xsdtest on them. See:</p> <p>"XML Schema for C++ Example 8: pub.xsd" on page 27-18</p> <p>"XML Schema for C++ Example 9: pub.xml" on page 27-20</p> <p>"XML Schema for C++ Example 10: pub.std" on page 27-21</p>

To build the sample programs, run **'make'**.

To build the programs and run them, comparing the actual output to expected output, run **'make sure'**.

## Error Messages are in English

An error message file is provided in the mesg subdirectory. Currently, the only message file is in English although message files for other languages may be supplied in future releases. You should set the environment variable `ORA_XML_MESG` to point to the absolute path of the mesg subdirectory.

Alternately, if you have an `$ORACLE_HOME` installed, you can copy the contents of the mesg subdirectory to the `$ORACLE_HOME/oracore/mesg` directory.

## XML Schema for C++ Example 1: xsdtest.cpp

```
// Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved.

//
// NAME validate.cpp
// DESCRIPTION Sample usage of C++ XML Schema processor
//

#include <iostream.h>
#include <string.h>
```

```
#ifndef ORAXML_CPP_ORACLE
include <oraxml.hpp>
#endif

#ifndef ORAXSD_CPP_ORACLE
include <oraxsd.hpp>
#endif

int main(int argc, char **argv)
{
 XMLSchema schema;
 XMLParser parser;
 xmlctx *ctx;
 char *doc, *uri;
 uword ecode;

 cout << "XML C++ Schema processor\n";

 if ((argc < 2) || (argc > 3))
 {
 cout << "usage: validate <xml document> [schema]\n";
 return -1;
 }
 doc = argv[1];
 uri = (argc > 2) ? argv[2] : 0;

 cout << "Initializing XML package...\n";

 if (ecode = parser.xmlinit())
 {
 cout << "Failed to initialize XML parser, error " << ecode;
 return 1;
 }

 cout << "Parsing '" << doc << "'...\n";
 if (ecode = parser.xmlparse((oraxtext *) doc, (oraxtext *) 0,
 XML_FLAG_DISCARD_WHITESPACE))
 {
 cout << "Parse failed, error " << ecode << "\n";
 return 2;
 }

 cout << "Initializing Schema package...\n";

 if (ecode = schema.initialize(&parser))
```

```

 {
 cout << "Failed, code " << ecode << "!\n";
 return 3;
 }

 cout << "Validating document...\n";
 if (ecode = schema.validate(&parser, (oratext *) uri))
 {
 cout << "Validation failed, error " << ecode << "\n";
 return 4;
 }

 cout << "Document is valid.\n";
 schema.terminate();
 return 0;
}

```

## XML Schema for C++ Example 2: car.xsd

```

<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
 targetNamespace = "http://www.CarDealers.com/">
 <element name="Car">
 <complexType>
 <element name="Model">
 <simpleType base="string">
 <enumeration value = "Ford"/>
 <enumeration value = "Saab"/>
 <enumeration value = "Audi"/>
 </simpleType>
 </element>
 <element name="Make">
 <simpleType base="string">
 <minLength value = "1"/>
 <maxLength value = "30"/>
 </simpleType>
 </element>
 <element name="Year">
 <complexType content="mixed">
 <attribute name="PreviouslyOwned" type="string"
 use="required"/>
 <attribute name="YearsOwned" type="integer"
 use="optional"/>
 </complexType>
 </element>
 </complexType>
 </element>

```

```

 </element>
 <element name="OwnerName" type="string"
 minOccurs="0" maxOccurs="unbounded"/>
 <element name="Condition">
 <complexType base="string" derivedBy="extension">
 <attribute name="Automatic">
 <simpleType base="string">
 <enumeration value = "Yes"/>
 <enumeration value = "No"/>
 </simpleType>
 </attribute>
 </complexType>
 </element>
 <element name="Mileage">
 <simpleType base="integer">
 <minInclusive value="0"/>
 <maxInclusive value="2000000"/>
 </simpleType>
 </element>
 <attribute name="RequestDate" type="date"/>
</complexType>
</element>
</schema>

```

## XML Schema for C++ Example 3: car.xml

```

<?xml version="1.0"?>
<car:Car xmlns:car="http://www.CarDealers.com/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xsi:schemaLocation="http://www.CarDealers.com/ car.xsd"
 RequestDate="2000-12-6">

 <Model>Ford</Model>
 <Make>Explorer</Make>
 <Year PreviouslyOwned="You betcha">1999</Year>
 <OwnerName>Joe Smith</OwnerName>
 <OwnerName>Bob Jones</OwnerName>
 <Condition Automatic="No">Small dent on right bumper.</Condition>
 <Mileage>1999999</Mileage>
</car:Car>

```

## XML Schema for C++ Example 4: car.std

XML C++ Schema processor

```

Initializing XML package...
Parsing 'car.xml'...
Initializing Schema package...
Validating document...
Document is valid.

```

## XML Schema for C++ Example 5: aq.xsd

```

<?xml version="1.0"?>
<!-- ***** AQ xml schema ***** -->
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
 targetNamespace = "http://www.oracle.com/AQXMLDocument"
 xmlns:aq = "http://www.oracle.com/AQXMLDocument"
 xmlns:xsd = "http://www.w3.org/1999/XMLSchema"
 elementFormDefault="qualified">

 <element name="AQXMLDocument">
 <complexType content="mixed">
 <choice>
 <group ref="aq:client_operation" minOccurs="0"/>
 <group ref="aq:server_response"/>
 </choice>
 </complexType>
 </element>

 <!-- ***** Client Operations Group ***** -->
 <group name="client_operation">
 <sequence>
 <element ref="aq:client_operation" minOccurs="0" maxOccurs="1"/>
 <choice>
 <element ref="aq:producer_options" maxOccurs="1"/>
 <element ref="aq:consumer_options" maxOccurs="1"/>
 <element ref="aq:register_options" maxOccurs="1"/>
 </choice>
 <element ref="aq:message_set" minOccurs="0" maxOccurs="*" />
 </sequence>
 </group>

 <!-- ***** Server Response Group ***** -->
 <group name="server_response">
 <sequence>
 <element ref="aq:server_response" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:receive_result" maxOccurs="1"/>
 <choice minOccurs="0" >
 <element ref="aq:send_result" maxOccurs="1"/>
 </choice>
 </sequence>
 </group>

```

```

 <element ref="aq:publish_result" maxOccurs="1"/>
 <element ref="aq:receive_result" maxOccurs="1"/>
 <element ref="aq:sequence_num_result" maxOccurs="1"/>
 </choice>
</sequence>
</group>

<!-- ***** Server Propagation Group ***** -->
<group name="server_prop_operation">
 <sequence>
 <element ref="aq:server_prop_operation" minOccurs="0" maxOccurs="1"/>
 <choice>
 <element ref="aq:push" maxOccurs="1"/>
 <element ref="aq:notification" maxOccurs="1"/>
 <element ref="aq:sequence_num_request" maxOccurs="1"/>
 </choice>
 </sequence>
</group>

<!-- ***** Client Operation ***** -->
<element name="client_operation">
 <complexType content="mixed">
 <element ref="aq:txid" minOccurs="0"/>
 </complexType>
<attribute name="opcode" use="required" type="aq:opcode_type"/>
</element>

<!-- ***** Server Response ***** -->
<element name="server_response">
 <complexType content="mixed">
 <element ref="aq:txid" minOccurs="0"/>
 <element ref="aq:status_response" minOccurs="1"/>
 </complexType>
<attribute name="opcode" use="required" type="aq:opcode_type"/>
</element>

<!-- ***** Server Propagation Operation ***** -->
<element name="server_prop_operation">
 <complexType content="mixed">
 <element ref="aq:txid" minOccurs="0"/>
 </complexType>
<attribute name="prop_opcode" use="required" type="aq:prop_opcode_type"/>
</element>

<element name="txid" type="string"/>

```

```

....

<!-- ***** Message payload ***** -->
<element name="message_payload">
 <complexType>
 <choice>
 <element ref="aq:jms_text_message" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:jms_map_message" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:jms_bytes_message" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:jms_object_message" minOccurs="0" maxOccurs="1"/>
 <any minOccurs="0" maxOccurs="*" processContents="skip"/>
 </choice>
 </complexType>
</element>

<!-- ***** User-defined properties ***** -->
<element name="user_properties">
 <complexType content="mixed">
 <element ref="aq:property" minOccurs="0" maxOccurs="*" />
 </complexType>
</element>

<!-- ***** Property ***** -->
<element name="property">
 <complexType content="mixed">
 <element ref="aq:name" minOccurs="1" maxOccurs="1"/>
 <element ref="aq:value" minOccurs="1" maxOccurs="1"/>
 <attribute name="property_type" type="aq:prop_type"/>
 </complexType>
</element>

<!-- ***** Status response ***** -->
<element name="status_response">
 <complexType content="mixed">
 <element ref="aq:acknowledge" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:status_code" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:error_code" minOccurs="0" maxOccurs="1"/>
 <element ref="aq:error_message" minOccurs="0" maxOccurs="1"/>
 </complexType>
</element>

<!-- ***** Send result ***** -->
<element name="send_result">
 <complexType content="mixed">
 <element ref="aq:destination" minOccurs="1" maxOccurs="1"/>
 </complexType>
</element>

```



```

 <element ref="aq:message_id" minOccurs="0" maxOccurs="*" />
 </complexType>
</element>

<!-- ***** Publish result ***** -->
<element name="publish_result">
 <complexType content="mixed">
 <element ref="aq:destination" minOccurs="1" maxOccurs="1" />
 <element ref="aq:message_id" minOccurs="0" maxOccurs="*" />
 </complexType>
</element>

<!-- ***** Receive result ***** -->
<element name="receive_result">
 <complexType content="mixed">
 <element ref="aq:destination" minOccurs="1" maxOccurs="1" />
 <element ref="aq:message_set" minOccurs="0" maxOccurs="*" />
 </complexType>
</element>

.
.

<!-- ***** JMS text message ***** -->
<element name="jms_text_message">
 <complexType content="mixed">
 <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1" />
 <element ref="aq:user_properties" minOccurs="0" maxOccurs="1" />
 <element ref="aq:text_data" minOccurs="1" maxOccurs="1" />
 </complexType>
</element>

<element name="text_data" type="string" />

<!-- ***** JMS map message ***** -->
<element name="jms_map_message">
 <complexType content="mixed">
 <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1" />
 <element ref="aq:user_properties" minOccurs="0" maxOccurs="1" />
 <element ref="aq:map_data" minOccurs="1" maxOccurs="1" />
 </complexType>
</element>

<!-- ***** Map data ***** -->
<element name="map_data">
 <complexType content="mixed">

```

```

 <element ref="aq:item" minOccurs="0" maxOccurs="*" />
 </complexType>
</element>

<!-- ***** Map Item ***** -->
<element name="item">
 <complexType content="mixed">
 <element ref="aq:name" minOccurs="1" maxOccurs="1" />
 <element ref="aq:value" minOccurs="1" maxOccurs="1" />
 </complexType>
</element>

<!-- ***** JMS bytes message ***** -->
<element name="jms_bytes_message">
 <complexType content="mixed">
 <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1" />
 <element ref="aq:user_properties" minOccurs="0" maxOccurs="1" />
 <element ref="aq:bytes_data" minOccurs="1" maxOccurs="1" />
 </complexType>
</element>

<element name="bytes_data" type="string" />

<!-- ***** JMS object message ***** -->
<element name="jms_object_message">
 <complexType content="mixed">
 <element ref="aq:oracle_jms_properties" minOccurs="0" maxOccurs="1" />
 <element ref="aq:user_properties" minOccurs="0" maxOccurs="1" />
 <element ref="aq:ser_object_data" minOccurs="1" maxOccurs="1" />
 </complexType>
</element>
<element name="ser_object_data" type="string" />

</schema>

```

## XML Schema for C++ Example 6: aq.xml

```

<AQXmlDocument xmlns="http://www.oracle.com/AQXmlDocument"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xsi:schemaLocation="http://www.oracle.com/AQXmlDocument aq.xsd">
 <client_operation opcode="SEND">
 <txid> sdasdfsdf </txid>
 </client_operation>

```

```
<producer_options delivery_mode="PERSISTENT">
 <destination lookup_type="NORMAL"> queue1 </destination>
 <priority>23</priority>
 <recipient_list>
 <recipient> abc </recipient>
 <recipient lookup_type="LDAP"> abc </recipient>
 </recipient_list>
</producer_options>

<message_set>
 <message_count>1</message_count>
 <message>
 <message_number>1</message_number>
 <message_header>
 <correlation>XML_40_NEW_TEST</correlation>
 <delay>10</delay>
 <sender_id>scott::home::0</sender_id>
 </message_header>
 <message_payload>
 <jms_map_message>
 <oracle_jms_properties>
 <reply_to>oracle::redwoodshores::100</reply_to>
 <userid>scott</userid>
 <appid>AQProduct</appid>
 <groupid>AQ</groupid>
 </oracle_jms_properties>
 <user_properties>
 <property property_type="STRING">
 <name>country</name>
 <value>USA</value>
 </property>
 <property property_type="STRING">
 <name>State</name>
 <value>california</value>
 </property>
 </user_properties>
 <map_data>
 <item item_type="STRING">
 <name>Car</name>
 <value>Toyota</value>
 </item>
 <item item_type="STRING">
 <name>Color</name>
 <value>Blue</value>
 </item>
 </map_data>
 </jms_map_message>
 </message_payload>
 </message>
</message_set>
```

```
 </item>
 <item item_type="STRING">
 <name>Shape</name>
 <value>Circle</value>
 </item>
 <item item_type="NUMBER">
 <name>Price</name>
 <value>20000</value>
 </item>
 </map_data>
</jms_map_message>
</message_payload>
</message>
</message_set>
</AQXmlDocument>
```

## XML Schema for C++ Example 7: aq.std

```
XML C++ Schema processor
Initializing XML package...
Parsing 'aq.xml'...
Initializing Schema package...
Validating document...
Document is valid.
```

## XML Schema for C++ Example 8: pub.xsd

```
<?xml version="1.0"?>
<schema xmlns = "http://www.w3.org/2000/08/XMLSchema"
 targetNamespace = "http://www.somewhere.org/BookCatalogue"
 xmlns:cat = "http://www.somewhere.org/BookCatalogue"
 elementFormDefault="qualified">
 <complexType name="Pub">
 <sequence>
 <element name="Title" type="cat:titleType" maxOccurs="*" />
 <element name="Author" type="string" maxOccurs="*" />
 <element name="Date" type="date" />
 </sequence>
 <attribute name="language" type="string" use="default" value="English" />
 <anyAttribute namespace="##local" />
 </complexType>
 <element name="Publication" type="cat:Pub" abstract="true" />
 <element name="Book" substitutionGroup="cat:Publication">
 <complexType>
 <complexContent>
```

```

 <extension base="cat:Pub" >
 <sequence>
 <element name="ISBN" type="string" default="123456789"/>
 <element name="Publisher" type="string"/>
 </sequence>
 </extension>
 </complexContent>
</complexType>
</element>
<complexType name="titleType">
 <simpleContent>
 <extension base="string" >
 <attribute name="old" type="string" use="default" value="false"/>
 </extension>
 </simpleContent>
</complexType>
<element name="Magazine" substitutionGroup="cat:Publication">
 <complexType>
 <complexContent>
 <extension base="cat:Pub">
 <sequence>
 <element name="Volume" type="cat:VolumeType"/>
 <element name="htmlTable">
 <complexType>
 <any namespace="##other"
 processContents="skip"
 minOccurs="0" maxOccurs="2"/>
 </complexType>
 </element>
 </sequence>
 </extension>
 </complexContent>
 </complexType>
</element>
<simpleType name="VolumeType">
 <restriction base="integer" >
 <minInclusive value = "1"/>
 <maxInclusive value = "12"/>
 </restriction>
</simpleType>
<element name="Catalogue">
 <complexType>
 <sequence>
 <element ref="cat:Publication" minOccurs="0" maxOccurs="*/>
 </sequence>
 </complexType>

```

```

 </complexType>
 </element>
</schema>

```

## XML Schema for C++ Example 9: pub.xml

```

<?xml version="1.0"?>
<Catalogue xmlns = "http://www.somewhere.org/BookCatalogue"
 xmlns:cat = "http://www.somewhere.org/BookCatalogue"
 xmlns:html = "http://www.somewhere.org/HTMLCatalogue"
 xmlns:xsi = "http://www.w3.org/1999/XMLSchema-instance"
 xsi:schemaLocation =
 "http://www.somewhere.org/BookCatalogue pub.xsd">
 <cat:Magazine>
 <Title>Natural Health</Title>
 <Author>October</Author>
 <Date>1999-12</Date>
 <Volume>12</Volume>
 <htmlTable>
 <table xmlns = "http://www.somewhere.org/HTMLCatalogue">
 <tr>...</tr>
 </table>
 <html:table>
 <html:tr>...</html:tr>
 </html:table>
 </htmlTable>
 </cat:Magazine>
 <Book>
 <Title>Illusions The Adventures of a Reluctant Messiah</Title>
 <Author>Richard Bach</Author>
 <Date>1977</Date>
 <ISBN></ISBN>
 <Publisher>Dell Publishing Co.</Publisher>
 </Book>
 <Book>
 <Title>The First and Last Freedom</Title>
 <Author>J. Krishnamurti</Author>
 <Date>1954</Date>
 <ISBN>0-06-064831-7</ISBN>
 <Publisher>Harper & Row</Publisher>
 </Book>
</Catalogue>

```

## XML Schema for C++ Example 10: pub.std

```
XML C++ Schema processor
Initializing XML package...
Parsing 'pub.xml'...
Initializing Schema package...
Validating document...
Document is valid.
```





---

## Using XML C++ Class Generator

This chapter contains the following sections:

- [Accessing XML C++ Class Generator](#)
- [Using XML C++ Class Generator](#)
- [XML C++ Class Generator Usage](#)
- [xmlcg Usage](#)
- [Using the XML C++ Class Generator Examples in sample/](#)

## Accessing XML C++ Class Generator

The XML C++ Class Generator is provided with Oracle and is also available for download from the OTN site: <http://otn.oracle.com/tech/xml>

It is located in `$ORACLE_HOME/xdk/cpp/classgen`.

## Using XML C++ Class Generator

The XML C++ Class Generator creates source files from an XML DTD. The Class Generator takes the Document Type Definition (DTD) and generates classes for each defined element. Those classes are then used in a C++ program to construct XML documents conforming to the DTD.

This is useful when an application wants to send an XML message to another application based on an agreed-upon DTD or as the back end of a web form to construct an XML document. Using these classes, C++ applications can construct, validate, and print XML documents that comply with the input DTD.

The Class Generator works in conjunction with the Oracle XML Parser for C++, which parses the DTD and passes the parsed document to the class generator.

**See Also:** [Chapter 3, "Oracle XML Developer Kits \(XDKs\) and Components: Overview and General FAQs"](#), under "Using Oracle XML Components to Generate XML Documents: C++" on page 3-22

## External DTD Parsing

The XML C++ Class Generator can also parse an external DTD directly without requiring a complete (dummy) document. By using the Oracle XML Parser for C++ routine, `xmlparsedtd()`.

The provided command-line program `xmlcg` has a new '-d' option that is used to parse external DTDs. See "[xmlcg Usage](#)" on page 28-4.

## Error Message Files

Error message files are provided in the `mesg/` subdirectory. The messages files also exist in the `$ORACLE_HOME/oracore/mesg` directory. You may set the environment variable `ORA_XML_MESG` to point to the absolute path of the `mesg/` subdirectory although this not required.

## XML C++ Class Generator Usage

Figure 28-1 summarizes the XML C++ Class Generator usage.

1. From the bin directory, at the command line, enter the following:

```
xml [XML document file name, such as xxxxxx]
```

where XML document file name is the name of the parsed XML document or parsed DTD being processed. The XML document must have an associated DTD.

The Input to the XML C++ Class Generator is an XML document containing a DTD, or an external DTD. The document body itself is ignored; only the DTD is relevant, though the document must conform to the DTD.

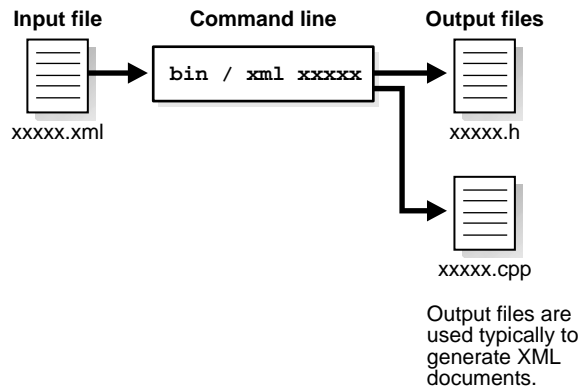
Accepted character set encoding for input files are listed in [Appendix F, "XDK for C++: Specifications and Cheat Sheet"](#).

2. Two source files are output, a xxxxx.h header file and a xxxxx.cpp C++ file. These are named after the DTD file.
3. The output files are typically used to generate XML documents.

Constructors are provided for each class (element) that allow an object to be created in the following two ways:

- Initially empty, then adding the children or data after the initial creation
- 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.

**Figure 28–1 XML C++ Class Generator Functionality**

## xmlcg Usage

The standalone parser may be called as an executable by invoking `bin/xmlcg`. For example:

```
xmlcg [flags] <XML document or External DTD>
```

[Table 28–1](#) lists the `xmlcg` optional flags.

**Table 28–1 xmlcg Optional Flags**

xmlcg Optional Flags	Description
-d name	DTD - Input is an external DTD with the given name
-o directory	Output directory for generated files (default is current directory)
-e encoding	Encoding - Default input file encoding
-h	Help - Show this usage help
-v	Version - Show the Class Generator version

## Using the XML C++ Class Generator Examples in sample/

[Table 28-2](#) lists the files supplied the sample XML C++ Class Generator sample/ directory.

**Table 28-2 XML C++ Class Generator Examples in sample/**

Sample File Name	Description
CG.cpp	Sample program
CG.xml	XML file contains DTD and dummy document
CG.dtd	DTD file referenced by CG.xml
Make.bat on Windows NT Makefile on UNIX	Batch file (on Windows NT) or script file (on UNIX) to generate classes and build the sample programs.
README	A readme file with these instructions

The `make.bat` batch file (on Windows NT) or `Makefile` (on UNIX) do the following:

- Generate classes based on `CG.xml` into `Sample.h` and `Sample.cpp`
- Compile the program `CG.cpp` (using `Sample.h`), and link this with the `Sample` object into an executable named `CG.exe` in the `...\bin` (or `.../bin`) directory.

### XML C++ Class Generator Example 1: XML — Input File to Class Generator, `CG.xml`

This XML file, `CG.xml`, inputs XML C++ Class Generator. It references the DTD file, `CG.dtd`.

```
<?xml version="1.0"?>
<!DOCTYPE Sample SYSTEM "CG.dtd">
 <Sample>
 Be!
 <D attr="value"></D>
 <E>
 <F>Formula1</F>
 <F>Formula2</F>
 </E>
 </Sample>
```

## XML C++ Class Generator Example 2: DTD — Input File to Class Generator, CG.dtd

This DTD file, CG.dtd is referenced by the XML file CG.xml. CG.xml inputs XML C++ Class Generator.

```
<!ELEMENT Sample (A | (B, (C | (D, E))) | F)>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA | F)*>
<!ELEMENT C (#PCDATA)>
<!ELEMENT D (#PCDATA)>
<!ATTLIST D attr CDATA #REQUIRED>
<!ELEMENT E (F, F)>
<!ELEMENT F (#PCDATA)>
```

## XML C++ Class Generator Example 3: CG Sample Program

The CG sample program, CG.cpp, does the following:

1. Initializes the XML parser
2. Loads the DTD (by parsing the DTD-containing file-- the dummy document part is ignored)
3. Creates some objects using the generated classes
4. Invokes the validation function which verifies that the constructed classes match the DTD
5. Writes the constructed document to Sample.xml

```
////////////////////////////////////
// NAME CG.cpp
// DESCRIPTION Demonstration program for C++ Class Generator usage
////////////////////////////////////

#ifdef ORAXMLDOM_ORACLE
include <oraxml.h>
#endif

#include <fstream.h>

#include "Sample.h"

#define DTD_DOCUMENT "CG.xml"
#define OUT_DOCUMENT "Sample.xml"
```

```
int main()
{
 XMLParser parser;
 Document *doc;
 Sample *samp;
 B *b;
 D *d;
 E *e;
 F *f1, *f2;
 fstream *out;
 ub4 flags = XML_FLAG_VALIDATE;
 uword ecode;

 // Initialize XML parser
 cout << "Initializing XML parser...\n";
 if (ecode = parser.xmlinit())
 {
 cout << "Failed to initialize parser, code " << ecode << "\n";
 return 1;
 }

 // Parse the document containing a DTD; parsing just a DTD is not
 // possible yet, so the file must contain a valid document (which
 // is parsed but we're ignoring).
 cout << "Loading DTD from " << DTD_DOCUMENT << "... \n";
 if (ecode = parser.xmlparse((oratext *) DTD_DOCUMENT, (oratext *)0, flags))
 {
 cout << "Failed to parse DTD document " << DTD_DOCUMENT <<
 ", code " << ecode << "\n";
 return 2;
 }

 // Fetch dummy document
 cout << "Fetching dummy document...\n";
 doc = parser.getDocument();

 // Create the constituent parts of a Sample
 cout << "Creating components...\n";
 b = new B(doc, (String) "Be there or be square");
 d = new D(doc, (String) "Dit dah");
 d->setattr((String) "attribute value");
 f1 = new F(doc, (String) "Formulal");
 f2 = new F(doc, (String) "Formula2");
 e = new E(doc, f1, f2);
}
```

```
 // Create the Sample
 cout << "Creating top-level element...\n";
 samp = new Sample(doc, b, d, e);

 // Validate the construct
 cout << "Validating...\n";
 if (ecode = parser.validate(samp))
 {
 cout << "Validation failed, code " << ecode << "\n";
 return 3;
 }

 // Write out doc
 cout << "Writing document to " << OUT_DOCUMENT << "\n";
 if (!(out = new fstream(OUT_DOCUMENT, ios::out)))
 {
 cout << "Failed to open output stream\n";
 return 4;
 }
 samp->print(out, 0);
 out->close();

 // Everything's OK
 cout << "Success.\n";

 // Shut down
 parser.xmlterm();
 return 0;
}

// end of CG.cpp
```



# Part X

---

## XDK for PL/SQL

Part X describes how to access and use Oracle XML Developer's Kit (XDK) for PL/SQL. It contains the following chapter:

- [Chapter 29, "Using XML Parser for PL/SQL"](#)

---

---

**Note:** XML-SQL Utility (XSU) for PL/SQL is considered part of the XDK for PL/SQL. In this manual, XSU is described in Part II, [Chapter 7, "XML SQL Utility \(XSU\)"](#).

---

---

FAQs are located at the end of Chapter 29, ["Frequently Asked Questions \(FAQs\): XML Parser for PL/SQL"](#) on page 29-20.



---

## Using XML Parser for PL/SQL

This chapter contains the following sections:

- [Accessing XML Parser for PL/SQL](#)
- [What's Needed to Run XML Parser for PL/SQL](#)
- [Using XML Parser for PL/SQL \(DOM Interface\)](#)
- [Using the XML Parser for PL/SQL: XSL-T Processor \(DOM Interface\)](#)
- [Using XML Parser for PL/SQL Examples in sample/](#)
- [Frequently Asked Questions \(FAQs\): XML Parser for PL/SQL](#)

## Accessing XML Parser for PL/SQL

XML Parser for PL/SQL is provided with Oracle and is also available for download from the OTN site: <http://otn.oracle.com/tech/xml>.

It is located at `$ORACLE_HOME/xdk/plsql/parser`

## What's Needed to Run XML Parser for PL/SQL

[Appendix G, "XDK for PL/SQL: Specifications and Cheat Sheets"](#) lists the specifications and requirements for running the XML Parser for PL/SQL. It also includes syntax cheat sheets.

## Using XML Parser for PL/SQL (DOM Interface)

The XML Parser for PL/SQL makes developing XML applications with Oracle a simplified and standardized process. With the PL/SQL interface, Oracle shops familiar with PL/SQL can extend existing applications to take advantage of XML as needed.

Since the XML Parser for PL/SQL is implemented in PL/SQL and Java, it can run "out of the box" on the Oracle Java Virtual Machine.

XML Parser for PL/SQL supports the W3C XML 1.0 specification. The goal is to be 100% conformant. It can be used both as a validating or non-validating parser.

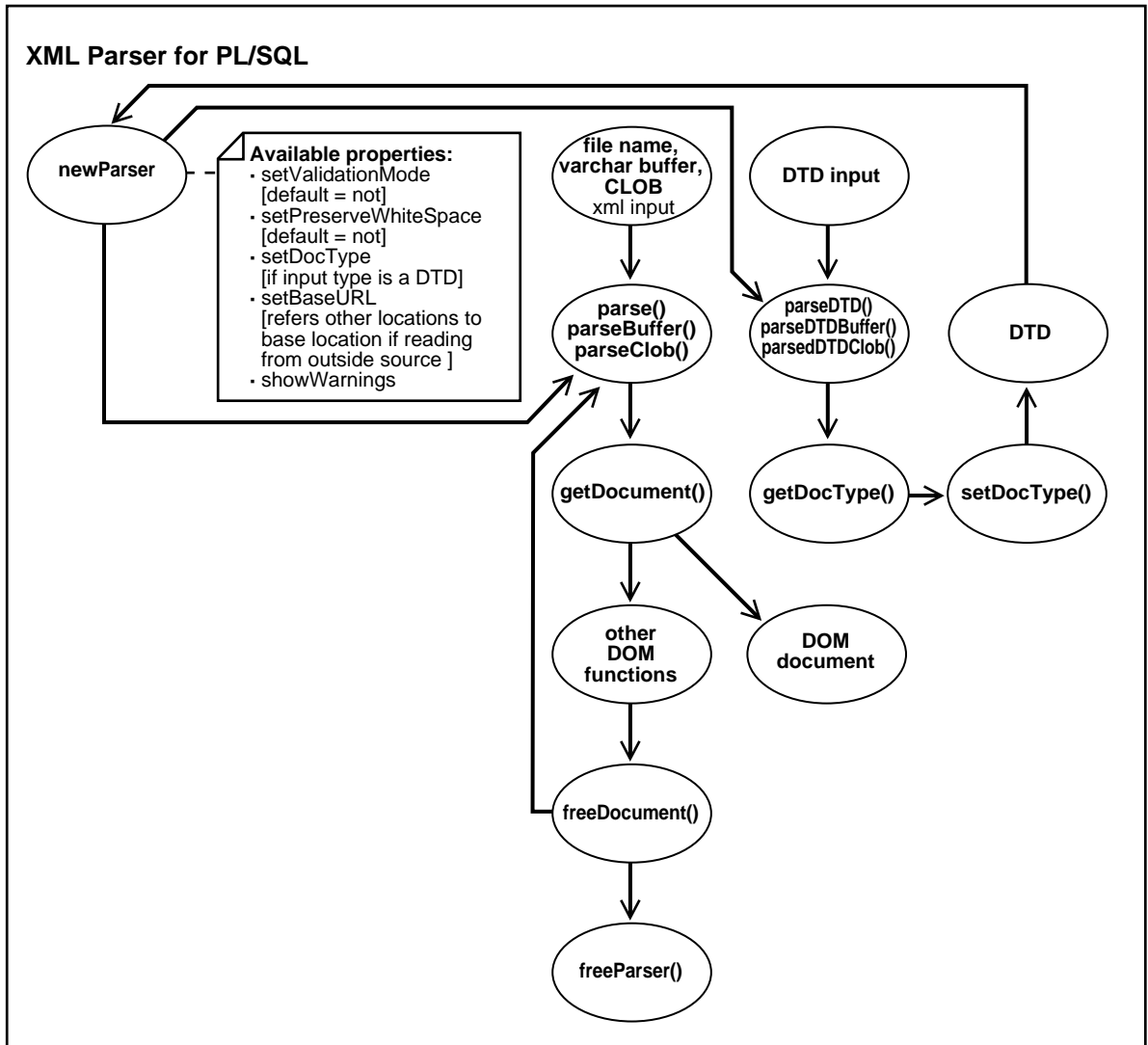
In addition, XML Parser for PL/SQL provides the two most common APIs you need for processing XML documents:

- W3C-recommended Document Object Model (DOM)
- XSL-T and XPath recommendations

This makes writing custom applications that process XML documents straightforward in the Oracle environment, and means that a standards-compliant XML parser is part of the Oracle platform on every operating system where Oracle is ported.

[Figure 29-1](#) shows the XML Parser for PL/SQL usage and parsing process diagram.

Figure 29–1 XML Parser for PL/SQL Functionality (DOM Interface)



1. Make a `newParser` declaration to begin the parsing process for the XML document and DTD, if applicable.

Table 29–1 lists available properties for the `newParser` procedure:

**Table 29–1 XML Parser for PL/SQL: newParser() Properties**

Property	Description
setValidationMode	Default = Not
setPreserveWhiteSpace	Default = Not
setDocType	Use if input type is a DTD
setBaseURL	Refers to other locations to the base locations, if reading from an outside source
showWarnings	Turns warnings on or off.

2. The XML and DTD can be input as a file, varchar buffer, or CLOB. The XML input is called by the following procedures:

- `parse()` if the XML input is a file
- `parseBuffer()` if the XML input is an varchar buffer
- `parserClob()` if the XML input is a CLOB

If a DTD is also input, it is called by the following procedures:

- `parseDTD()` if the input is an DTD file
- `parseDTDBuffer()` if the DTD input is an varchar buffer
- `parserDTDClob()` if the DTD input is a CLOB

**For the XML Input:** For an XML input, the parsed result from `Parse()`, `ParserBuffer()`, or `ParserClob()` procedures is sent to `GetDocument()`.

3. `getDocument()` procedure performs the following:

- Outputs the parsed XML document as a DOM document typically to be used in a PL/SQL application, or
- Applies other DOM functions, if applicable.

**See Also:** *Oracle9i XML Reference* for a list of available optional DOM functions.

4. Use `freeDocument()` function to free up the parser and parse the next XML input
5. Use `freeParser()` to free up any temporary document structures created during the parsing process

**For the DTD input:** The parsed result from `parseDTD()`, `parseDTDBuffer()`, or `parseDTDClob()` is used by `getDocType()` function.

6. `getDocType()` then uses `setDocType()` to generate a DTD object.
7. The DTD object can be fed into the parser using `setDocType()` to override the associated DTD.

## XML Parser for PL/SQL: Default Behavior

The following is the default behavior for XML Parser for PLSQL XML:

- 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

The types and methods described in this manual are supplied with the PLSQL package `xmlparser()`.

## Using the XML Parser for PL/SQL: XSL-T Processor (DOM Interface)

Extensible Stylesheet Language Transformation, abbreviated XSLT (or XSL-T), 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 PLSQL implementation of the XSL processor follows the W3C XSLT working draft (rev WD-xslt-19990813) and includes the required behavior of an XSL processor in terms of how it must read XSLT stylesheets and the transformations it must effect.

The types and methods described in this document are made available by the PLSQL package, `xslprocessor()`.

[Figure 29-2](#) shows the XML Parser for PL/SQL XSL-T Processor main functionality.

1. The Process Stylesheet process receives input from the XML document and the selected Stylesheet, which may or may not be indicated in the XML document. Both the stylesheet and XML document can be the following types:

- File name
- Varchar buffer
- CLOB

The XML document can be input 1 through n times.

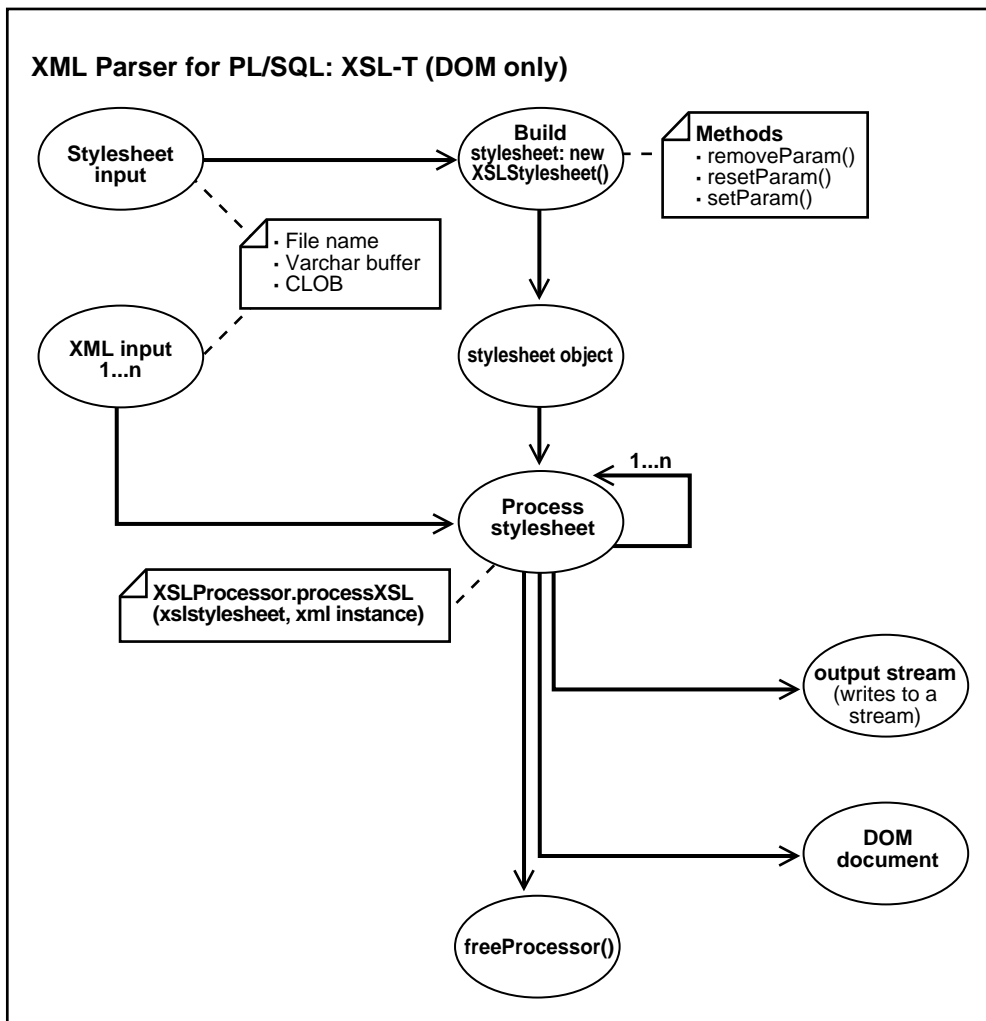
2. The parsed XML document inputs  
`XSLProcessor.processXSL(xslstylesheet,xml instance)`  
procedure, where:
  - XML document is indicated in the "xml instance" argument
  - Stylesheet input is indicated in the "xslstylesheet" argument
3. Build the stylesheet using the Stylesheet input to the `XSLStylesheet()` procedure. The following methods are available for this procedure:
  - `removeParam()`
  - `resetParam()`
  - `setParam()`



This produces a stylesheet object which then inputs the "Process Stylesheet" step using procedure, `XSLProcessor.processXSL(xslstylesheet,xml instance)`.

4. The "Process stylesheet" process can be repeated 1 through n times. In other words, the same stylesheet can be applied to multiple parsed XML documents to transform them wither into an XML document, HTML document, or other text based format.
5. The resulting parsed and transformed document is output either as a stream or a DOM document.
6. When the XSL-T process if complete, call the `freeProcessor()` procedure to free up any temporary structures and the `XSLProcessor` procedures used in the XSL transformation process.

Figure 29-2 "XML Parser for PL/SQL: XSL-T processor (DOM Interface)



## XML Parser for PL/SQL: XSLT Processor — Default Behavior

The following is the default behavior for the XML Parser for PL/SQL XSLT 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

## Using XML Parser for PL/SQL Examples in sample/

### Setting Up the Environment to Run the sample/ Sample Programs

The `$ORACLE_HOME/xdk/plsql/parser/sample/` directory contains two sample XML applications:

- domsample
- xlsample

These show you how to use XML Parser for PL/SQL.

To run these sample programs carry out the following steps:

1. Load the PL/SQL parser into the database. To do this, follow the instructions given in the README file under the lib directory.
2. You must have the appropriate java security privileges to read and write from a file on the file system. To this, first startup SQL\*Plus (located typically under `$ORACLE_HOME/bin`) and connect as a user with administration privileges, such as, 'internal':

For example

```
% sqlplus
SQL> connect / as sysdba
```

3. A password might be required or the appropriate user with administration privileges. Contact your System Administrator, DBA, or Oracle support, if you cannot login with administration privileges.
4. Give special privileges to the user running this sample. It must be the same one under which you loaded the jar files and plsql files in Step 1.

For example, for user 'scott':

```
SQL> grant javauserpriv to scott;
SQL> grant javasyspriv to scott;
```

You should see two messages that say "Grant succeeded." Contact your System Administrator, DBA, or Oracle support, if this does not occur.

Now, connect again as the user under which the PL/SQL parser was loaded in step 1. For example, for user 'scott' with password 'tiger':

```
SQL> connect scott/tiger
```

## Running domsample

To run domsample carry out the following steps:

1. Load domsample.sql script under SQL\*Plus (if SQL\*Plus is not up, first start it up, connecting as the user running this sample) as follows:

```
SQL> @domsample
```

The domsample.sql script defines a procedure domsample with the following syntax:

```
domsample(dir varchar2, infile varchar2, errfile varchar2)
```

where:

Argument	Description
'dir'	Must point to a valid directory on the external file system and should be specified as a complete path name
'infile'	Must point to the file located under 'dir', containing the XML document to be parsed
'errfile'	Must point to a file you wish to use to record errors; this file will be created under 'dir'

2. Execute the domsample procedure inside SQL\*Plus by supplying appropriate arguments for 'dir', 'infile', and 'errfile'. For example:

On Unix, you can could do the following:

```
SQL>execute domsample('/private/scott', 'family.xml', 'errors.txt');
```

On Windows NT, you can do the following:

```
SQL>execute domsample('c:\xml\sample', 'family.xml', 'errors.txt');
```

where family.xml is provided as a test case

3. You should see the following output:
  - The elements are: family member member member member

- The attributes of each element are:

```
family:
lastname = Smith
 member:
 memberid = m1
 member:
 memberid = m2
 member:
 memberid = m3 mom = m1 dad = m2
 member:
 memberid = m4 mom = m1 dad = m2
```

## Running xslsample

To run xslsample, carry out these steps:

1. Load the xslsample.sql script under SQL\*Plus (if SQL\*Plus is not up, first start it up, connecting as the user running this sample):

```
SQL>@xslsample
```

xslsample.sql script defines a procedure xslsample with the following syntax:

```
xslsample (dir varchar2, xmlfile varchar2, xslfile varchar2, resfile
varchar2, errfile varchar2)
```

where:

Argument	Description
'dir'	Must point to a valid directory on the external file system and should be specified as a complete path name.
'xmlfile'	Must point to the file located under 'dir', containing the XML document to be parsed.
'xskfile'	Must point to the file located under 'dir', containing the XSL stylesheet to be applied.
'resfile'	Must point to the file located under 'dir' where the transformed document is to be placed.

Argument	Description
'errfile'	Must point to a file you wish to use to record errors; this file will be created under 'dir'

- Execute the `xslsample` procedure inside SQL\*Plus by supplying appropriate arguments for 'dir', 'xmlfile', 'xslfile', and 'errfile'.

For example:

- On Unix, you can do the following:

```
SQL>execute xslsample('/private/scott', 'family.xml', 'iden.xml',
'family.out', 'errors.txt');
```

- On NT, you can do the following:

```
SQL>execute xslsample('c:\xml\sample', 'family.xml', 'iden.xml',
'family.out', 'errors.txt');
```

- The provided test cases are: `family.xml` and `iden.xml`
- You should see the following output:

```
Parsing XML document c:\family.xml
Parsing XSL document c:\iden.xml
XSL Root element information
Qualified Name: xsl:stylesheet
Local Name: stylesheet
Namespace: http://www.w3.org/XSL/Transform/1.0
Expanded Name: http://www.w3.org/XSL/Transform/1.0:stylesheet
A total of 1 XSL instructions were found in the stylesheet
Processing XSL stylesheet
Writing transformed document
```

- `family.out` should contain the following:

```
<family lastname="Smith">
<member memberId="m1">Sarah</member>
<member memberId="m2">Bob</member>
<member memberId="m3" mom="m1" dad="m2">Joanne</member>
<member memberId="m4" mom="m1" dad="m2">Jim</member>
</family>
```

You might see a delay in getting the output when executing the procedure for the first time. This is because Oracle JVM performs various initialization tasks

before it can execute a Java Stored Procedure (JSP). Subsequent invocations should run quickly.

If you get errors, ensure the directory name is specified as a complete path on the file system

---



---

**Note:** SQL directory aliases and shared directory syntax '\\\ are not supported at this time.

---



---

Otherwise, report the problem on the XML discussion forum at <http://otn.oracle.com>

## XML Parser for PL/SQL Example 1: XML — family.xml

This XML file inputs `domsample.sql`.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE family SYSTEM "family.dtd">
<family lastname="Smith">
 <member memberid="m1">Sarah</member>
 <member memberid="m2">Bob</member>
 <member memberid="m3" mom="m1" dad="m2">Joanne</member>
 <member memberid="m4" mom="m1" dad="m2">Jim</member>
</family>
```

## XML Parser for PL/SQL Example 2: DTD — family.dtd

This DTD file is referenced by XML file, `family.xml`.

```
<!ELEMENT family (member*)>
<!ATTLIST family lastname CDATA #REQUIRED>
<!ELEMENT member (#PCDATA)>
<!ATTLIST member memberid ID #REQUIRED>
<!ATTLIST member dad IDREF #IMPLIED>
<!ATTLIST member mom IDREF #IMPLIED>
```

## XML Parser for PL/SQL Example 3: XSL — iden.xsl

This XSL file inputs the `xslsample.sql`.

```
<?xml version="1.0"?>
```

```
<!-- Identity transformation -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:template match="*|*|comment()|processing-instruction()|text()">
 <xsl:copy>
 <xsl:apply-templates select="*|*|comment()|processing-instruction()|text()"/>
 </xsl:copy>
 </xsl:template>
</xsl:stylesheet>
```

## XML Parser for PL/SQL Example 4: PL/SQL — domsample.sql

```
-- This file demonstrates a simple use of the parser and DOM API.
-- The XML file that is given to the application is parsed and the
-- elements and attributes in the document are printed.
-- It shows you how to set the parser options.

set serveroutput on;
create or replace procedure domsample(dir varchar2, infile varchar2,
 errfile varchar2) is

p xmlparser.parser;
doc xmldom.DOMDocument;

-- prints elements in a document
procedure printElements(doc xmldom.DOMDocument) is
nl xmldom.DOMNodeList;
len number;
n xmldom.DOMNode;

begin
 -- get all elements
 nl := xmldom.getElementsByTagName(doc, '*');
 len := xmldom.getLength(nl);

 -- loop through elements
 for i in 0..len-1 loop
 n := xmldom.item(nl, i);
 dbms_output.put(xmldom.getNodeName(n) || ' ');
 end loop;

 dbms_output.put_line('');
end printElements;

-- prints the attributes of each element in a document
procedure printElementAttributes(doc xmldom.DOMDocument) is
```



```
nl xmldom.DOMNodeList;
len1 number;
len2 number;
n xmldom.DOMNode;
e xmldom.DOMELEMENT;
nrm xmldom.DOMNamedNodeMap;
attrname varchar2(100);
attrval varchar2(100);

begin

 -- get all elements
 nl := xmldom.getElementsByTagName(doc, '*');
 len1 := xmldom.getLength(nl);

 -- loop through elements
 for j in 0..len1-1 loop
 n := xmldom.item(nl, j);
 e := xmldom.makeElement(n);
 dbms_output.put_line(xmldom.getTagName(e) || ':');

 -- get all attributes of element
 nrm := xmldom.getAttributes(n);

 if (xmldom.isNull(nrm) = FALSE) then
 len2 := xmldom.getLength(nrm);

 -- loop through attributes
 for i in 0..len2-1 loop
 n := xmldom.item(nrm, i);
 attrname := xmldom.getNodeName(n);
 attrval := xmldom.getNodeValue(n);
 dbms_output.put(' ' || attrname || ' = ' || attrval);
 end loop;
 dbms_output.put_line('');
 end if;
 end loop;

 end printElementAttributes;

begin

 -- new parser
 p := xmlparser.newParser;
```

```
-- set some characteristics
xmlparser.setValidationMode(p, FALSE);
xmlparser.setErrorLog(p, dir || '/' || errfile);
xmlparser.setBaseDir(p, dir);

-- parse input file
xmlparser.parse(p, dir || '/' || infile);

-- get document
doc := xmlparser.getDocument(p);

-- Print document elements
dbms_output.put('The elements are: ');
printElements(doc);

-- Print document element attributes
dbms_output.put_line('The attributes of each element are: ');
printElementAttributes(doc);

-- deal with exceptions
exception

when xmldom.INDEX_SIZE_ERR then
 raise_application_error(-20120, 'Index Size error');

when xmldom.DOMSTRING_SIZE_ERR then
 raise_application_error(-20120, 'String Size error');

when xmldom.HIERARCHY_REQUEST_ERR then
 raise_application_error(-20120, 'Hierarchy request error');

when xmldom.WRONG_DOCUMENT_ERR then
 raise_application_error(-20120, 'Wrong doc error');

when xmldom.INVALID_CHARACTER_ERR then
 raise_application_error(-20120, 'Invalid Char error');

when xmldom.NO_DATA_ALLOWED_ERR then
 raise_application_error(-20120, 'Nod data allowed error');

when xmldom.NO_MODIFICATION_ALLOWED_ERR then
 raise_application_error(-20120, 'No mod allowed error');

when xmldom.NOT_FOUND_ERR then
 raise_application_error(-20120, 'Not found error');
```

```

when xmldom.NOT_SUPPORTED_ERR then
 raise_application_error(-20120, 'Not supported error');

when xmldom.INUSE_ATTRIBUTE_ERR then
 raise_application_error(-20120, 'In use attr error');

end domsample;
/
show errors;

```

## XML Parser for PL/SQL Example 5: PL/SQL — xslsample.sql

```

-- This file demonstrates a simple use of XSL-T transformation capabilities.
-- The XML and XSL files that are given to the application are parsed,
-- the transformation specified is applied and the transformed document is
-- written to a specified result file.
-- It shows you how to set the parser options.

```

```

set serveroutput on;
create or replace procedure xslsample(dir varchar2, xmlfile varchar2,
 xslfile varchar2, resfile varchar2,
 errfile varchar2) is

```

```

p xmlparser.Parser;
xmldoc xmldom.DOMDocument;
xmldocnode xmldom.DOMNode;
proc xslprocessor.Processor;
ss xslprocessor.Stylesheet;
xsl doc xmldom.DOMDocument;
docfrag xmldom.DOMDocumentFragment;
docfragnode xmldom.DOMNode;
xslelem xmldom.DOMELEMENT;
nspc varchar2(50);
xslcmds xmldom.DOMNodeList;

begin

-- new parser
p := xmlparser.newParser;

-- set some characteristics
xmlparser.setValidationMode(p, FALSE);
xmlparser.setErrorLog(p, dir || '/' || errfile);
xmlparser.setPreserveWhiteSpace(p, TRUE);
xmlparser.setBaseDir(p, dir);

```

```
-- parse xml file
 dbms_output.put_line('Parsing XML document ' || dir || '/' || xmlfile);
 xmlparser.parse(p, dir || '/' || xmlfile);

-- get document
 xmldoc := xmlparser.getDocument(p);

-- parse xsl file
 dbms_output.put_line('Parsing XSL document ' || dir || '/' || xslfile);
 xmlparser.parse(p, dir || '/' || xslfile);

-- get document
 xsldoc := xmlparser.getDocument(p);

 xslelem := xmldom.getDocumentElement(xsldoc);
 namespace := xmldom.getNamespace(xslelem);

-- print out some information about the stylesheet
 dbms_output.put_line('XSL Root element information');
 dbms_output.put_line('Qualified Name: ' ||
 xmldom.getQualifiedName(xslelem));
 dbms_output.put_line('Local Name: ' ||
 xmldom.getLocalName(xslelem));
 dbms_output.put_line('Namespace: ' || namespace);
 dbms_output.put_line('Expanded Name: ' ||
 xmldom.getExpandedName(xslelem));

 xslcmds := xmldom.getChildrenByTagName(xslelem, '*', namespace);
 dbms_output.put_line('A total of ' || xmldom.getLength(xslcmds) ||
 ' XSL instructions were found in the stylesheet');

-- make stylesheet
 ss := xslprocessor.newStylesheet(xsldoc, dir || '/' || xslfile);

-- process xsl
 proc := xslprocessor.newProcessor;
 xslprocessor.showWarnings(proc, true);
 xslprocessor.setErrorLog(proc, dir || '/' || errfile);

 dbms_output.put_line('Processing XSL stylesheet');
 docfrag := xslprocessor.processXSL(proc, ss, xmldoc);
 docfragnode := xmldom.makeNode(docfrag);

 dbms_output.put_line('Writing transformed document');
 xmldom.writeToFile(docfragnode, dir || '/' || resfile);
```

```
-- deal with exceptions
exception

when xmldom.INDEX_SIZE_ERR then
 raise_application_error(-20120, 'Index Size error');

when xmldom.DOMSTRING_SIZE_ERR then
 raise_application_error(-20120, 'String Size error');

when xmldom.HIERARCHY_REQUEST_ERR then
 raise_application_error(-20120, 'Hierarchy request error');

when xmldom.WRONG_DOCUMENT_ERR then
 raise_application_error(-20120, 'Wrong doc error');

when xmldom.INVALID_CHARACTER_ERR then
 raise_application_error(-20120, 'Invalid Char error');

when xmldom.NO_DATA_ALLOWED_ERR then
 raise_application_error(-20120, 'Nod data allowed error');

when xmldom.NO_MODIFICATION_ALLOWED_ERR then
 raise_application_error(-20120, 'No mod allowed error');

when xmldom.NOT_FOUND_ERR then
 raise_application_error(-20120, 'Not found error');

when xmldom.NOT_SUPPORTED_ERR then
 raise_application_error(-20120, 'Not supported error');

when xmldom.INUSE_ATTRIBUTE_ERR then
 raise_application_error(-20120, 'In use attr error');

end xslsample;
/
show errors;
```

## Frequently Asked Questions (FAQs): XML Parser for PL/SQL

### Exception in Thread Parser Error

#### Question

When I try to use the `oraxsl` I get the following: Exception in thread "main":

```
java.lang.NoClassDefFoundError: oracle/xml/parser/v2/oraxsl.
```

How do I fix this?

#### Answer

Can you provide more details as to your configuration and usage? If you are running outside the database you need to make sure the `xmlparserv2.jar` is explicitly in your `CLASS_PATH` not simply its directory. If from the database you need to make sure it has been properly loaded and that JServer initialized.

### Encoding '8859\_1' is not currently supported by the JavaVM?

#### Question

I parsed my XML document using the XML Parser for PL/SQL and modified some of the node values of the `DOMDocument` by using `setNodeValue`. When I tried to write the modified `DOMDocument` to buffer or file using `write To Buffer` or `ratatouille`, both commands gave me the following error:

```
ORA-20101: Error occurred while accessing a file or URL: Encoding '8859_1' is not currently supported by the JavaVM
```

#### Comment

I just reinstalled `initjvm.sql` and also installed the latest version of the XML Parser for PL/SQL. Everything is working fine.

### `xmlDom.GetNodeValue` in PL/SQL

#### Question

I cannot get the element value using the PL/SQL `XMLDOM`. Here is the code fragment:

```

..nl := xmldom.getElementsByTagName(doc, '*');
len := xmldom.getLength(nl)
;-- loop through elements
 for i in 0..len-1 loop n := xmldom.item(nl, i);
 elename := xmldom.getNodeName(n);
 elevel := xmldom.getNodeValue(n);
 ..elename is Ok, but elevel is NULL.

```

Associating with a text node does not seem to work, or I am not doing it correctly? I receive a compile error, for example:

```

...t xmldom.DOMText;
...t := xmldom.makeText(n);
elevel := xmldom.getNodeValue(t);

```

What am I doing wrong?

### Comment

I found the answer to my own question. To get the text node value associated with the element node, you must perform additional node navigation via `xmldom.getFirstChild(n)`.

To illustrate, change `printElements()` in `DOMSample.sql` as follows:

```

begin
-- get all elements
nl := xmldom.getElementsByTagName(doc, '*');
 len := xmldom.getLength(nl);
 -- loop through elements
 for i in 0..len-1 loop n := xmldom.item(nl, i);
 dbms_output.put(xmldom.getNodeName(n));
 -- get the text node associated with the element node
 n := xmldom.getFirstChild(n);
 if xmldom.getNodeType(n) = xmldom.TEXT_NODE then dbms_
output.put('=' | | xmldom.getNodeValue(n));
 end if;
 dbms_output.put(' ');
 end loop;
 dbms_output.put_line('');
end printElements;

```

This produces the following output:

The elements are:

```
family member=Sarah member=Bob member=Joanne member=Jim
```

The attributes of each element are:

```
family:familylastname val=Smithmember:membermemberid val=m1member:membermemberid
val=m2member:membermemberid val=m3 mom val=m1 dad val=m2member:membermemberid
val=m4 mom val=m1 dad val=m2
```

## XDK for PL/SQL Toolkit

### Question

I downloaded XDK for PL/SQL but it requires OAS. Do you have any idea how to run this in an IIS environment?

### Answer

If you're going to use IIS, it would be better to use the XML Parser for Java V2. You'll need Oracle.

## Parsing DTD contained in a CLOB (PL/SQL) XML

### Question

I am having problems parsing a DTD file contained in a CLOB. I used the API, "xmlparser.parseDTDClob", provided by the XML Parser for PL/SQL.

The following error was thrown:

```
"ORA-29531: no method parseDTD in class oracle/xml/parser/plsql/XMLParserCover".
```

The procedure `xmlparser.parseDTDClob` calls a Java Stored Procedure `xmlparsercover.parseDTDClob`, which in turn calls another Java Stored Procedure `xmlparsercover.parseDTD`.

I have confirmed that the class file, "oracle.xml.parser.plsql.XMLParserCover", has been loaded into the database, and that it has been published. So the error message does not make sense. The procedure used to call "xmlparser.parseDTDClob" is:

```
create or replace procedure parse_my_dtd as p xmlparser.parser; l_clob clob;
begin p := xmlparser.newParser; select content into l_clob from dca_
documents where doc_id = 1; xmlparser.parseDTDClob(p,l_clob,'site_template');
end; API Documentation for xmlparser.parseDTDClob:
```

```
parseDTDClob PURPOSE Parses the DTD stored in the given clob SYNTAX
```



```
PROCEDURE parseDTDClob(p Parser, dtd CLOB, root VARCHAR2); PARAMETERS p
(IN)- parser instance dtd (IN)- dtd clob to parse root (IN)- name
of the root element RETURNS Nothing COMMENTS
```

Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing failed, for some reason.

Description of the table dca\_documents:

DOC_ID	NOT NULL	NUMBER	DOC_NAME	NOT NULL	VARCHAR2(350)	DOC_
TYPE			VARCHAR2(30)			
DESCRIPTION			VARCHAR2(4000)	MIME_TYPE		
VARCHAR2(48)	CONTENT	NOT NULL	CLOB	CREATED_BY	NOT NULL	
VARCHAR2(30)	CREATED_ON	NOT NULL	DATE	UPDATED_BY	NOT NULL	
VARCHAR2(30)	UPDATED_ON	NOT NULL	DATE			

The contents of the DTD:

```
<!ELEMENT site_template (component*)> <!ATTLIST site_template template_id CDATA
#REQUIRED> <!ATTLIST site_template template_name CDATA #REQUIRED> <!ELEMENT
component (#PCDATA)> <!ATTLIST component component_id ID #REQUIRED> <!ATTLIST
component parent_id ID #REQUIRED> <!ATTLIST component component_name ID
#REQUIRED>
```

## Answer

This is a known issue in the 1.0.1 release of the XML Parser for PL/SQL. Here is the workaround.

1. Make a backup of ./plsxmlparser\_1.0.1/lib/sql/xmlparsercover.sql
2. In line 18 in xmlparsercover.sql, change the string:  
oracle.xml.parser.plsql.XMLParserCover.parseDTD to  
oracle.xml.parser.plsql.XMLParserCover.parseDTDClob
3. Verify that Line 18 now reads: procedure parseDTDClob(id varchar2, DTD  
CLOB, root varchar2, err in out varchar2) is language java name  
'oracle.xml.parser.plsql.XMLParserCover.parseDTDClob(java.lang.String,  
oracle.sql.CLOB, java.lang.String, java.lang.String[])';
4. Save the file
5. Rerun xmlparsercover.sql in SQL\*Plus Assuming you've loaded XMLParserV2  
release 2.0.2.6 into the database, this should solve your problem.

## XML Parser for PL/SQL

### Question

I have just started using XML Parser for PL/SQL. I am have trouble getting the text between the begin tag and the end tag into a local variable. Do you have examples?

### Answer

You just have to use the following:

```
selectSingleNode("pattern");
getNodeValue()
```

Remember, if you are trying to get value from a Element node, you have to move down to the #text child node, for example, `getFirstChild().getNodeValue()`

Suppose you need to get the text contained between the starting and ending tags of a `xml.dom.DOMNode n`. The following 2 lines will suffice.

```
n_child:=xml.dom.getFirstChild(n);
text_value:=xml.dom.getNodeValue(n_child);
```

`n_child` is of type `xml.dom.DOMNode`

`text_value` is of type `varchar2`

## Security: ORA-29532, Granting JavaSysPriv to User

### Question

We are using the XML Parser for PLSQL and are trying to parse an XML document. We are getting a Java security error:

```
ORA-29532: Java call terminated by uncaught Java exception:
java.lang.SecurityException ORA-06512: at "NSEC.XMLPARSERCOVER", line 0
ORA-06512: at "NSEC.XMLPARSER", line 79 ORA-06512: at "NSEC.TEST1_XML line 36
ORA-06512: at line 5
```

Do we need to grant to user? The syntax appears correct. We also get the error when we run the demo.

## Answer

If the document you are parsing contains a `<!DOCTYPE` which has a System URI with a protocol like `file:///` or `http:///` then you to grant an appropriate privilege to your current database user to be able to "reach out of the database", so to speak, and open a stream on the file or URL.`CONNECT SYSTEM/MANAGER`

```
GRANT JAVAUSERPRIV, JAVASYSPRIV TO youruser;
```

should do it.

## Installing XML Parser for PL/SQL: JServer(JVM) Option

### Question

I have downloaded and installed the `plxmlparser_V1_0_1.tar.gz`. The readme said to use `loadjava` to upload `xmlparserv2.jar` and `plsql.jar` in order. I tried to load `xmlparserv2.jar` using the following command:

```
loadjava -user test/test -r -v xmlparserv2.jar
```

to upload the jar file into Oracle8i. After much of the uploading, I got the following error messages:

```
identical: oracle/xml/parser/v2/XMLConstants is unchanged from previously loaded
fileidentical: org/xml/sax/Locator is unchanged from previously loaded
fileloading : META-INF/MANIFEST.MFcreating : META-INF/MANIFEST.MFError while
creating resource META-INF/MANIFEST.MF ORA-29547: Java system class not
available: oracle/aurora/rdbms/Compilerloading :
oracle/xml/parser/v2/msg/XMLErrorMsg_en_US.propertiescreating :
oracle/xml/parser/v2/msg/XMLErrorMsg_en_US.propertiesError while creating
...
```

Then I removed `-r` from the previous command:

```
loadjava -user test/test -v xmlparserv2.jar
```

I still got errors but it's down to four:

```
.identical: org/xml/sax/Locator is unchanged from previously loaded fileloading
: META-INF/MANIFEST.MFcreating : META-INF/MANIFEST.MFError while creating
...
```

I think I have installed the JServer on the database, correctly.

### Answer

The JServer option is not properly installed if you're getting errors like this during loadjava. You need to run INITJVM.SQL and INITDBJ.SQL to get the JavaVM properly installed. Usually these are in the ./javavm subdirectory of your Oracle Home.

## XML Parser for PL/SQL: domsample

### Question

I am trying to execute domsample on dom1151. This is an example that is provided with installation. XML file family.xml is present in the directory /hrwork/log/pqpd115CM/out.

Still I am getting the following error.

Usage of domsample is domsample(dir, inpfile, errfile)

```
SQL>
begin
domsample('/hrwork/log/pqpd115CM/out', 'family.xml', 'errors.txt');
end;
/
Error generated :
begin
*
ERROR at line 1:
ORA-20100: Error occurred while parsing: No such file or directory
ORA-06512: at "APPS.XMLPARSER", line 22
ORA-06512: at "APPS.XMLPARSER", line 69
ORA-06512: at "APPS.DOMSAMPLE", line 80
ORA-06512: at line 2
```

### Answer

From your description it sounds like you have not completed all of the steps in the sample/Readme without errors. After confirming the xmlparserv2.jar is loaded, carefully complete the steps again.

## XML in CLOBs

### Question

In Oracle8i database, we have CLOBs which contain well formed XML documents up to 1 MB in size.

We want the ability to extract only part of the CLOB (XML document), modify it, and replace it back in the database rather than processing the entire document.

Second, we want this process to run entirely on the database tier.

Which products or tools are needed for this? This may be possible with the JVM which comes with Oracle. There also may be some PL/SQL tools available to achieve this by means of stored procedures.

### Answer

You can do this by using either of the following:

- Oracle XML Parser for PLSQL
- Create your own custom Java stored procedure wrappers over some code you write yourselves with the Oracle XML Parser for Java.

XML Parser for PLSQL has methods like:

- `xmlparser.parseCLOB()`

As well as methods like:

- `xslProcessor.selectNodes()` to find what part of the doc you are looking for
- `xmlDOM.*` methods to manipulate the content of the XML Doc
- `xmlDOM.writeToCLOB()` to write it back

If you wanted to do surgical updates on the text of the CLOB, you would have to use `DBMS_LOB.*` routines, but this would be tricky unless the changes being made to the content don't involve any growth or shrinkage of the number of characters.

## Out of memory errors in `oracle.xml.parser`

### Question

Out of memory errors in `oracle.xml.parser`

last entry at 2000-04-26 10:59:27.042:

```
VisiBroker for Java runtime caught exception:
java.lang.OutOfMemoryError
 at oracle.xml.parser.v2.XMLAttrList.put(XMLAttrList.java:251)
 at oracle.xml.parser.v2.XMLElement.setAttribute(XMLElement.java:260)
 at oracle.xml.parser.v2.XMLElement.setAttribute(XMLElement.java:228)
 at cars.XMLServer.processEXL(XMLServer.java:122)
```

It's trying to create a new XML attribute and crashes with `OutOfMemoryError`.

We are parsing a 50Mb XML file. We have upped the `java_pool_size` to 150Mb with a `shared_pool_size` of 200Mb.

### Answer

You should not be using the `DOMParser` for parsing a 50Mb XML file. You need to look at the `SAXParser` which parses files of arbitrary size because it does not create an in-memory tree of nodes as it goes.

Which parser are you using, SAX or DOM - if you are using DOM, you should seriously consider moving to SAX which processes the XML file sequentially instead of trying to build an in-memory tree that represents the file.

Using SAX we process XML files in excess of 180Mb without any problems and with very low memory requirements.

Rule of thumb for DOM and SAX:

DOM:

- DOM is very good when you need some sort of random access
- DOM consumes more memory
- DOM is also good when you are trying to transformations of some sort
- DOM is also good when you want to have tree iteration and want to walk through the entire document tree
- See if you can use more attributes over elements in your XML (to reduce the pipe size)

SAX:

- SAX is good when data comes in a streaming manner (using some input stream)

## Is There a PL/SQL Parser Based on C?

### Question

Is there a PL/SQL parser that is based on C?

### Answer

There is not one currently but there are plans to provide the PL/SQL parser on top of the C version.

## Memory Requirements When Using the Parser for PL/SQL

### Question

What are the memory requirements for using the PL/SQL Parser?

### Answer

While the memory use is directly dependent on the document size, it should also be realized that the PL/SQL parser uses the Java parser and thus the Oracle JServer is being run. JServer typically requires 40-60MB depending on its configuration.

## JServer (JVM), Is It Needed to Run XML Parser for PL/SQL?

### Question

Do I need to install JServer to run the XML Parser for PL/SQL?

### Answer

Yes, if you are running the parser in the database, you do need JServer because the PL/SQL Parser currently uses the XML Parser for Java under the covers. JServer exists in both the Standard and Enterprise versions. A forthcoming version of XML Parser for PL/SQL using C underneath is being developed for applications that do not have access to a Java Virtual Machine (JVM).

## Using the DOM API

### **What does the XML Parser for PL/SQL do?**

The XML parser accepts any XML document giving you a tree-based API (DOM) to access or modify the document's elements and attributes. It also supports XSLT which allows transformation from one XML document to another.

### **Question - Is it possible to dynamically set the encoding in the XML document?**

No, you need to include the proper encoding declaration in your document as per the specification. You cannot use `setCharset(DOMDocument)` to set the encoding for the input of your document. `SetCharset(DOMDocument)` is used with `oracle.xml.parser.v2.XMLDocument` to set the correct encoding for the printing.

### **Question - How do I get the number of elements in a particular tag using the parser?**

You can use the `getElementByTagName` (elem `DOMElement`, name IN `VARCHAR2`) method that returns a `DOMNodeList` of all descent elements with a given tag name. You can then find out the number of elements in that `DOMNodeList` to determine the number of the elements in the particular tag.

### **Question - How do I parse a string?**

We do not currently have any method that can directly parse an XML document contained within a `String`. You can use

- function `parse` (Parser, `VARCHAR2`) to parse XML data stored in the given URL or the given file,
- function `parseBuffer` (Parser, `VARCHAR2`) to parse XML data stored in the given buffer, or
- function `parseCLOB` (Parser, `VARCHAR2`) to parse XML data stored in the give CLOB.

### **Question - How do I display my XML document?**

If you are using IE5 as your browser you can display the XML document directly. Otherwise, you can use our XSLT processor in v2 of the parser to create the HTML document using an XSL Stylesheet. Our Java Transviewer bean also allows you to view your XML document.



**Question - How do I write the XML data back using a special character sets?**

You can specify the character sets for writing to a file or a buffer. Writing to a CLOB will be using the default character set for the database that you are writing to. Here are the methods to use:

- procedure writeToFile(doc DOMDocument, fileName VARCHAR2, charset VARCHAR2);
- procedure writeToBuffer(doc DOMDocument, buffer IN OUT VARCHAR2, charset VARCHAR2);
- procedure writeToClob(doc DOMDocument, cl IN OUT CLOB, charset VARCHAR2);

**Question - How do I to get ampersand from characterData?**

You cannot have "raw" ampersands in XML data. You need to use the entity, &amp; instead. This is defined in the XML standard.

**Question - How do I generate a document object from a file?**

Check out the following example:

```
inpPath VARCHAR2;
inpFile VARCHAR2;
p xmlparser.parser;
doc xmldom.DOMDocument;

-- initialize a new parser object;
p := xmlparser.newParser;
-- parse the file
xmlparser.parse(p, inpPath || inpFile);
-- generate a document object
doc := xmlparser.getDocument(p);
```

**Question - Can the parser run on Linux?**

As long as a 1.1.x or 1.2.x JavaVM for Linux exists in your installation, you can run the Oracle XML Parser for Java there. Otherwise, you can use the C or C++ XML Parser for Linux.

**Question - How do I perform a >,<,>=, or <= comparison using the XML Parser v2?**

You need to use the entities &lt; for < and &gt; for >.

**Question -Is support for Namespaces and Schema included?**

The current XML parsers support Namespaces. Schema support will be included in a future release.

**Question -My parser doesn't find the DTD file.**

The DTD file defined in the <!DOCTYPE> declaration must be relative to the location of the input XML document. Otherwise, you'll need to use the `setBaseDir(Parser, VARCHAR2)` functions to set the base URL to resolve the relative address of the DTD.

**Question - Can I validate an XML file using an external DTD?**

You need to include a reference to the applicable DTD in your XML document. Without it there is no way that the parser knows what to validate against. Including the reference is the XML standard way of specifying an external DTD. Otherwise you need to embed the DTD in your XML Document.

**Question - Do you have DTD caching?**

Yes, DTD caching is optional and it is not enabled automatically.

**Question - How do I get the DOCTYPE tag into the XMLDocument after its parsed?**

You need to do some preprocessing to the file, and then put it through the `DOMParser` again, which will produce a valid, well-formed `XMLDocument` with the `DOCTYPE` tag contained within.

**Question - How does the XML DOM parser work?**

The parser accepts an XML formatted document and constructs in memory a DOM tree based on its structure. It will then check whether the document is well-formed and optionally whether it complies with a DTD. It also provides methods to traverse the tree and return data from it.

**Question - How do I create a node whose value I can set later?**

If you check the DOM spec referring to the table discussing the node type, you will find that if you are creating an element node, its `nodeValue` is to be null and hence cannot be set. However, you can create a text node and append it to the element node. You can store the value in the text node.

**Question - How do I extract elements from the XML file?**

If you're using DOM, the you can use the NamedNodeMap methods to get the elements.

**Question - How do I append a text node to a DOMELEMENT using PL/SQL parser?**

Use the createTextNode() method to create a new text node. Then convert the DOMELEMENT to a DOMNode using makeNode(). Now, you can use appendChild() to append the text node to the DOMELEMENT.

**Question - I am using XML parser with DOM but I cannot get the actual data. What is wrong?**

You need to check at which level your data resides. For example,

- `<?xml version=1.0 ?>`
- `<greeting>Hello World!</greeting>`

The text is the first child node of the first DOM element in the document. According to the DOM Level 1 spec, the "value" of an ELEMENT node is null and the getNodeValue() method will always return null for an ELEMENT type node. You have to get the TEXT children of an element and then use the getNodeValue() method to retrieve the actual text from the nodes.

**Question - Can the XML Parser for PL/SQL handle stylesheets that produce non-XML documents such as HTML?**

Yes it can.

## Using the Sample

**Question - I cannot run the sample file. Did I do something wrong in the installation?**

Here are two frequently missing steps in installing the PL/SQL parser:

- initialize the JServer -- run `$ORACLE_HOME/javavm/install/initjvm.sql`
- load the included jar files from the parser archive.

## XML Parser for PL/SQL: Parsing DTD in a CLOB

### Question

I am having problems parsing a DTD file contained in a CLOB. I used the API, "xmlparser.parseDTDClob", provided by the XML Parser for PL/SQL.

The following error was thrown:

```
"ORA-29531: no method parseDTD in class oracle/xml/parser/plsql/XMLParserCover"
```

I managed to work out the following:

The procedure `xmlparser.parseDTDClob` calls a Java Stored Procedure `xmlparsercover.parseDTDClob`, which in turn calls another Java Stored Procedure `xmlparsercover.parseDTD`.

I have confirmed that the class file -"oracle.xml.parser.plsql.XMLParserCover" has been loaded into the database, and that it has been published. So the error message does not make sense.

I am not able to figure out whether I am doing it right or whether this is a bug in the parser API.

The procedure use to call "xmlparser.parseDTDClob" :

```

create or replace procedure parse_my_dtd as
p xmlparser.parser;
l_clob clob;
begin
 p := xmlparser.newParser;
 select content into l_clob from dca_documents where doc_id = 1;
 xmlparser.parseDTDClob(p,l_clob,'site_template');
end;
```

### API Documentation for `xmlparser.parseDTDClob`:

`parseDTDClob`

PURPOSE

Parses the DTD stored in the given clob

SYNTAX

```
PROCEDURE parseDTDClob(p Parser, dtd CLOB, root VARCHAR2);
```

PARAMETERS

```
p (IN)- parser instance
dtd (IN)- dtd clob to parse
root (IN)- name of the root element
```

RETURNS  
Nothing  
COMMENTS

Any changes to the default parser behavior should be effected before calling this procedure. An application error is raised if parsing failed, for some reason.

Description of the table `dca_documents`:

<code>DOC_ID</code>	NOT NULL	NUMBER
<code>DOC_NAME</code>	NOT NULL	VARCHAR2(350)
<code>DOC_TYPE</code>		VARCHAR2(30)
<code>DESCRIPTION</code>		VARCHAR2(4000)
<code>MIME_TYPE</code>		VARCHAR2(48)
<code>CONTENT</code>	NOT NULL	CLOB
<code>CREATED_BY</code>	NOT NULL	VARCHAR2(30)
<code>CREATED_ON</code>	NOT NULL	DATE
<code>UPDATED_BY</code>	NOT NULL	VARCHAR2(30)
<code>UPDATED_ON</code>	NOT NULL	DATE

The contents of the DTD:

```
<!ELEMENT site_template (component*)>
<!ATTLIST site_template template_id CDATA #REQUIRED>
<!ATTLIST site_template template_name CDATA #REQUIRED>
<!ELEMENT component (#PCDATA)>
<!ATTLIST component component_id ID #REQUIRED>
<!ATTLIST component parent_id ID #REQUIRED>
<!ATTLIST component component_name ID #REQUIRED>
```

### Answer (a)

It appears to be a typo in the "xmlparsercover.sql" script which is defining the Java Stored Procedures that wrap the XMLParser. It mentions the Java method name "parseDTD" in the 'is language java name' part when "parseDTD" should be "parseDTDClob" (case-sensitive).

If you:

1. Make a backup copy of this script
2. Edit the line that reads:

```
procedure parseDTDClob(id varchar2,
dtd CLOB, root varchar2, err in out varchar2) is language java name
'oracle.xml.parser.plsql.XMLParserCover.parseDTD (java.lang.String,
oracle.sql.CLOB, java.lang.String, java.lang.String[])';
```

to say:

```
procedure parseDTDCLob(id varchar2,
 dtd CLOB, root varchar2, err in out varchar2) is language java name
 'oracle.xml.parser.plsql.XMLParserCover.parseDTDCLob
 (java.lang.String, oracle.sql.CLOB, java.lang.String,
 java.lang.String[])';
```

that is, change the string:

```
'oracle.xml.parser.plsql.XMLParserCover.parseDTD
```

to

```
'oracle.xml.parser.plsql.XMLParserCover.parseDTDCLob
```

and rerun the `xmlparsercover.sql` script you should be in business.

I filed a bug 1147031 to get this typo corrected in a future release.

Note: Your DTD had syntactic errors in it, but I was able to run the following without problem after making the change above:

```
declare
 c clob;
 v varchar2(400) :=
 '<!ELEMENT site_template (component*)>
 <!ATTLIST site_template template_name CDATA #IMPLIED
 template_id CDATA #IMPLIED >
 <!ELEMENT component (#PCDATA)>
 <!ATTLIST component component_id ID #REQUIRED
 parent_id IDREF #IMPLIED
 component_name CDATA #IMPLIED >';
begin
 delete from dca_documents;
 insert into dca_documents values(1,empty_clob())
 returning content into c;
 dbms_lob.writeappend(c,length(v),v);
 commit;
 parse_my_dtd;
end;
```

### Answer (b)

What do you want to do with the LOB? The LOB can either be a temporary LOB or a persistent LOB. In case of persistent lobs, you need to insert the value into a table. In case of temp LOB you can instantiate it in your program.

For example:

```
persistant lob
```

```

declare
 clob_var CLOB;
begin
 insert into tab_xxx values(EMPTY_CLOB()) RETURNING clob_col INTO
clob_var;
 dbms_lob.write(,,,);
 // send to AQ
end;
temp lob -----

declare
 a clob;
begin
 dbms_lob.createtemporary(a,DBMS_LOB.SESSION);
 dbms_lob.write(...);
 // send to AQ

end;
/

```

Also refer to *Oracle9i Application Developer's Guide - Large Objects (LOBs)*. There are 6 books (in PDF) one for each language access (C(OCI), Java, PL/SQL, Visual Basic, Pro\*C/C++, Pro\*Cobol) and it is quite comprehensive. If this is PL/SQL, I believe you can just do the following:

```
myClob CLOB = clob();
```

I have tried the `DBMS_LOB.createtemporary()` which works.

### Answer (c)

Here's what you need to do if you are using LOBs with AQ:

1. Create an ADT with one of the fields of type CLOB.

```
create type myAdt (id NUMBER, cdata CLOB);
```

The queue table must be declared to be of type myAdt

2. Instantiate the object - use `empty_clob()` to fill the LOB field

```
myMessage := myAdt(10, EMPTY_CLOB());
```

3. Enqueue the message

```

clob_loc clob;
enq_msgid RAW(16);
DBMS_AQ.enqueue('queue1', enq_opt, msg_prop, myMessage, enq_msgid)

```

4. Get the LOB locator

```
select t.user_data.cdata into clob_loc
from qtable t where t.msgid
= enq_msgid;
```

5. Populate the CLOB using dbms\_lob.write

6. Commit

There is an example of this is in the *Oracle8i Application Developer's Guide - Advanced Queuing*. If you are using the Java API for AQ, the procedure is slightly more complicated.

## Errors When Parsing a Document

I downloaded the javaparser v2 and the xml parser utility and I'm using the PLSQL parser interface. I have an XML file that is a composite of three tags and when parsing it generates the following error:

```
ORA-20100: Error occurred while parsing: Unterminated string
```

When I separate the document into individual tags 2 are ok the third generates this error:

```
ORA-20100: Error occurred while parsing: Invalid UTF8 encoding
```

1. Why is the error different when separating the data?
2. I have not been able to find an "unterminated string" in the document.
3. I'm fairly anxious since this is the only way the data is coming and I don't have time to figure out another parser.

### Answer

If you document is the "composite of three tags" then it is not a well-formed document as it has more than one root element. Try putting a start and end tag around the three.

## PLXML: Parsing a Given URL?

### Question

I am working with the XML parser for PL/SQL on NT. According to your Parser API documentation it is possible to parse a given url, too:> Parses xml stored in the



given url/file and returns> the built DOM DocumentNow, parsing from file works fine, but any form of url raises ORA-29532:... java.io.FileNotFoundException.

Can you give an example of a call?

### Answer

To access external URLs, you need set up your proxy host and port. For example using this type of syntax:

```
java -Dhttp.proxyHost=myproxy.mydomain.com -Dhttp.proxyPort=3182DOMSample
myxml.xml
```

## Using XML Parser to Parse HTML?

### Question

We need to parse HTML files as follows:

1. Find each "a href"
2. For each a href found, extract the file/pathname being linked to
3. Substitute a database procedure call for the a href, passing the file/pathname as a parameter.

Does it make sense to use the PL/SQL XML parser to do this? If so, how easy/hard would it be, and how can we find out how to do this?

### Answer

Since HTML files aren't necessary well formed XML documents, are you sure you want to use XML parser? Won't PERL be a better choice? I'm not sure whether PL/SQL parser supports the following methods but just for your information:

1. `getElementsByTagName()` retrieves all matching nodes.
2. `getNodeValue()` will return a string.
3. `setNodeValue()` sets node values.

### Answer b

It supports those methods, but not over an ill-formed HTML file.

## Oracle 7.3.4: Moving Data to a Web Browser (PL/SQL)

### Question

I'm trying to get the data to a web browser in the client side while all the processing has to take place on the server (oracle 7.3.4), using:

- XML Parser for PL/SQL
- XSQL servlet

Are these two components sufficient to get the job done?

### Answer

Dependencies for XSQL Page Processor states:

- Oracle XML Parser V2 R2.0.2.5
- Oracle XML-SQL Utility for Java
- Web server supporting Java Servlets
- JDBC driver

You'll also need XSQL Page Processor itself.

## Oracle 7.3.4 and XML

### Question

Does the XML Parser for Java,V2, work with Oracle 7.3.4.?

Is XML- SQL Utility part of XML Parser for Java,V2, or does it need to be downloaded separately.

### Answer

1. The XML Parser for Java, V2 works with 7.3.4 as long as you have the proper JDBC driver and run it in a VM on a middle tier or client.
2. The XML-SQL Utility includes a copy of the v2 parser in its download, as it requires it.

## getNodeValue(): Getting the Value of DomNode

### Question

I am having problems obtaining the value between XML tags after using `xmlparser()`. Below is code from the DOMSAMPLE.SQL example:

```
-- loop through elementsfor i in 0..len-1 loop n := xmlparser.item(nl, i);
 dbms_output.put(xmlparser.getNodeName(n))
```

### Comment

I encountered the same problem. I found out that `getNodeValue()` on Element Node returns null. `getNodeValue()` on the *text* node returns the value.

## Retrieving all Children or Grandchildren of a Node

### Question

Is there a way to retrieve all children or grandchildren, and so on, of a particular node in a DOM tree using the DOM API? Or is there a work-around? We are using the XML Parser for PL/SQL.

### Answer

Try the following:

```
DECLARE nodeList xmldom.DOMNodeList;
theElement xmldom.DOMELEMENT;
BEGIN :nodeList := xmldom.getElementsByTagName(theElement, '*');
:END;
```

This gets all children nodes rooted as the element in "theElement".

## What Causes ora-29532 "Uncaught java exception:java.lang.ClassCastException?"

### Question

We want to parse XML, apply XSL, and get the transformed result in the form of an XML document. We are using XML Parser for PL/SQL. Our script does not add PI instruction `<?xml version="1.0"?>` to the transformed result.

`XSLProcessor.processXSL` method returns documentfragment object.

Create DOMdocument object from that documentfragment object using: `finaldoc := xmldom.MakeDocument(docfragnode);`

Write to result file using where finaldoc is created of type `xmldom.DOMDocument`:

```
xmldom.writeToFile(finaldoc, dir || '/' || resfile);
```

This method is available for `DOMDocument`, but we are getting:

```
ora-29532 "Uncaught java exception: java.lang.ClassCastException"
```

I am not sure if converting documentfragment to domdocument object adds instruction "`<?xml version="1.0"?>`", or must we add this instruction through XSL?

### **Answer**

If you have created a new `DOMDocument` and then appended the document fragment to it, then you can use `xmldom.WriteToBuffer()` or similar routine to serialize with the XML declaration in place.

# A

---

## An XML Primer

This Appendix contains the following sections:

- [What is XML?](#)
- [W3C XML Recommendations](#)
- [XML Features](#)
- [How XML Differs From HTML](#)
- [Presenting XML Using Stylesheets](#)
- [Extensibility and Document Type Definitions \(DTD\)](#)
- [Why Use XML?](#)
- [Additional XML Resources](#)

## What is XML?

XML, eXtensible Markup Language, is the standard way to identify and describe data on the web. It is widely implementable and easy to deploy.

XML is a human-readable, machine-understandable, general syntax for describing hierarchical data, applicable to a wide range of applications, databases, e-commerce, Java, web development, searching, and so on.

Custom tags enable the definition, transmission, validation, and interpretation of data between applications and between organizations.

### Tag

XML elements use start tags (<) and end tags (>). For example, <author> where author, the name of the tag, is enclosed is start and end tags. You can name tags whatever you want.

### Attributes

Attributes add more information about each XML element. Attributes can be used to describe how the data is encoded or represented, to indicate where the links or external resources are located, to identify and call external processes such as applets, servlets, and so on, and to specify element instance in documents so that you can find them rapidly during a document search. Attributes also can provide extra information about the XML document's content or other elements. Attributes are not used to specify fonts, colors, or other style or formatting.

XML attributes can be held in the start tag of a start-end tag pair, or an empty tag. They can be name value pairs. For example, <image="adx10.jpg" ada\_txt="XSQL Description"/>. Attributes must always be in quotes.

Attributes and their content are defined in DTDs or XML Schema.

### Element

An example of an element in an XML document is <author>charles kopman</author>. The element includes the start, tag, end tag, and text in the middle of the start and end tags.

Every XML document must have a root or top-level element. This is the outermost element and contains all the other elements. You can select any name for your root element. In HTML, the root element was always <html>....</html>.

## Entity

Entities are virtual storage units that can contain graphics, text, sound files, binary data. In XML entities are represented by character strings. You can create your own entities. Five internal entities are already defined for you to use in XML:

less than sign, < uses &lt;

greater than sign, > uses &gt;

ampersand &, uses &amp;

single quote or apostrophe, ' uses &apos;

double quotation mark, " uses &quot;

## Basic Rules for XML Markup

Here are eight basic XML markup rules:

- Declare XML first. The very first line of your XML document must have an XML declaration that states that the XML document complies with the W3C XML recommendation. For example, `<?xml version="1.0" standalone="yes" ?>`
- Use one top-level tag, or "document element" or root tag. All tags and XML content are contained in (under) this top-level tag.
- Every element must have a start and end tag, for example, `<author>charles kopman</author>`
- Empty elements must end `/>`.  
For example, `<author name="charles kopman" />`
- Ensure that your elements are well nested in the correct hierarchy.
- All attribute values must be quoted with single or double quotes. For example, `<author name = "charles kopman">`
- Every XML tag begins with <. Every XML entity begins with & and ends with ;
- Remember the five internal entities, listed in the previous paragraph. See "Entity".
- You can tell the XML Parser which character encoding you are using in the XML declaration at the top of your XML document. For example: `<?xml version="1.0" encoding="ISO-8859-9" ?>`

For a comprehensive list of encoding, see:

<http://www.isi.edu/in-notes/iana/assignments/character-sets>

## W3C XML Recommendations

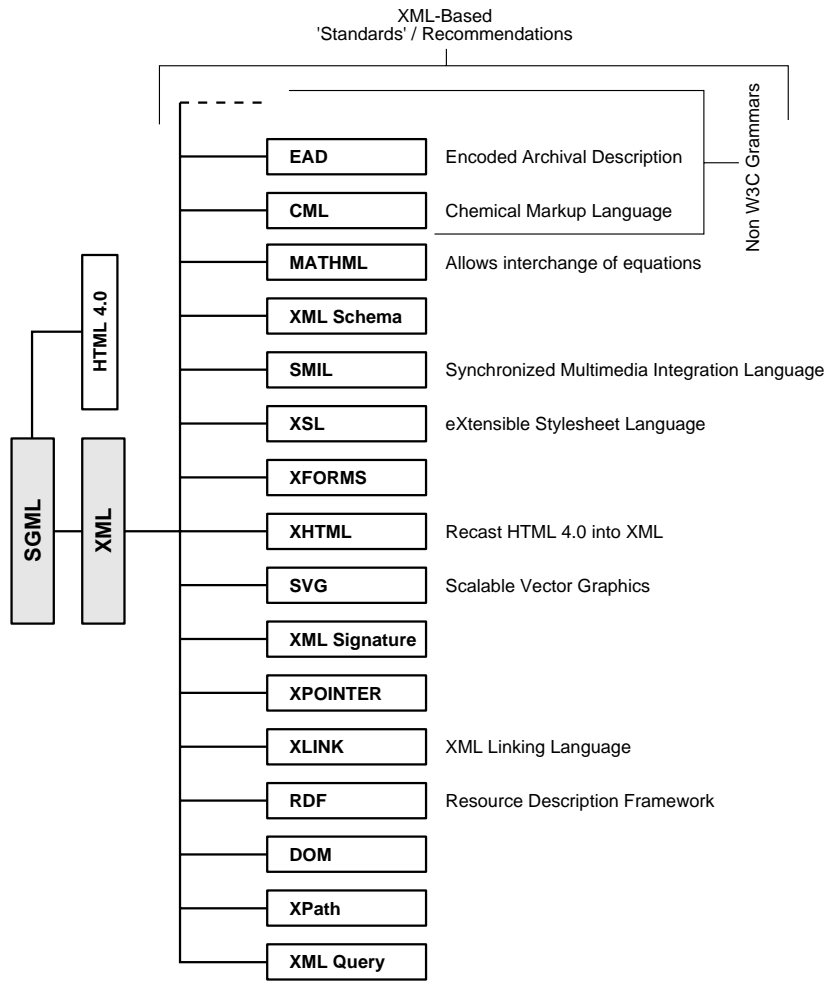
The World Wide Web Consortium (W3C) XML recommendations are an ever-growing set of interlocking specifications.

- **XML 1.0** was recommended by W3C in February 1998. It has resulted numerous additional W3C Working Groups, a Java Platform Extension Expert Group, and the XML conversion of numerous data interchange standards such as Electronic Data Interchange (EDI). The next version of HTML will be an XML application known as xHTML.
- **XML Namespaces**. Another W3C recommendation aimed at removing element ambiguity in multi-namespace well-formed XML applications.
- **XML Query**. The W3C standards effort to specify a query language for XML documents.
- **XML Schema**. The W3C standards effort to add simple and complex datatypes to XML documents and replace the functionality of DTDs with an XML Schema definition XML document.
- **XSL**. XSL consists of two W3C recommendations:
  - XSL Transformations for transforming one XML document into another
  - XSL Formatting Objects for specifying the presentation of an XML document
- **XPath**. XPath is the W3C recommendation that specifies the data model and grammar for navigating an XML document utilized by XSL-T, XLink, and XML Query.
- **XPointer**. XPointer is the W3C recommendation that specifies the identification of individual entities or fragments within an XML document using XPath navigation. This W3C proposed recommendation is defined at <http://www.w3.org/TR/WD-xptr>
- **DOM**. The W3C recommendation that specifies the Document Object Model of an XML Document including APIs for programmatic access.

The XML family of applications is illustrated in [Figure A-1](#).



Figure A-1 The XML Family of Applications ('Including XML-Based Standards')



## XML Features

The following bullets describe XML features:

- **Data Exchange, From Structured to Unstructured Data:** XML enables a universal standard syntax for exchanging data. XML specifies a rigorous, text-based way to represent the structure inherent in data so that it can be authored and interpreted unambiguously. Its simple, tag-based approach leverages developers' familiarity of HTML but provides a flexible, extensible mechanism that can handle the gamut of "digital assets" from highly structured database records to unstructured documents and everything in between. " W3C
- **SGML Was Designed Specifically for Documents - XML is Designed for Potentially Any Data:** The SGML markup language was specifically designed for documents. Web-centric XML is like a toolkit that can be used to write other languages. It is not designed for documents only. Any data that can be described in a tree can be programed in XML.
- **A Class of Data Objects - A Restricted Form of SGML:** [www.oasis-open.org](http://www.oasis-open.org) describes XML as follows: "... XML, describes a class of data objects called XML *documents* and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language. By construction, XML documents are conforming SGML documents."
- **XML's Many Uses...:** A W3C.org press release describes XML as follows: "... XML is primarily intended to meet the requirements of large-scale Web content providers for industry-specific markup, vendor-neutral data exchange, media-independent publishing, one-on-one marketing, workflow management in collaborative authoring environments, and the processing of Web documents by intelligent clients.
- **Metadata.** XML is also finding use in certain metadata applications.
- **Internationalization.** "XML is fully internationalized for both European and Asian languages, with all conforming processors required to support the Unicode character set in both its UTF-8 and UTF-16 encoding..." Its primary use is for electronic publishing and data interchange..."
- **Parsed or Unparsed Storage Entities:** From the W3C.org XML specification proposal: "... 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 the character data in the document, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure.

- **XML Processor Reads XML Documents.** "... 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...."
- **Open Internet Standard.** XML is gaining wide industry support from other vendors besides, like IBM, Sun, Microsoft, Netscape, SAP, CISCO and others, as a platform- and application-neutral format for exchanging information.

Although this manual is not intended to expound on XML syntax, a brief overview of some key XML topics is presented here. You can refer to the many excellent resources listed in "[Additional XML Resources](#)" for more information on XML syntax.

## How XML Differs From HTML

Like HTML, XML is a subset of SGML (Structured Generalized Markup Language), optimized for delivery over the web.

Unlike HTML, which tags elements in web pages for presentation by a browser, for example, `<bold>Oracle</bold>`, XML tags elements as data, such as, `<company>Oracle</company>`. You can use XML to give context to words and values in web pages, identifying them as data instead of simple textual or numeric elements.

The following example is in HTML code. This is followed by the corresponding XML example. The examples show employee data:

- Employee number
- Name
- Job
- Salary

### HTML Example 1

```
<table>
 <tr><td>EMPNO</td><td>ENAME</td><td>JOB</td><td>SAL</td></tr>
 <tr><td>7654</td><td>MARTIN</td><td>SALESMAN</td><td>1250</td></tr>
 <tr><td>7788</td><td>SCOTT</td><td>ANALYST</td><td>3000</td></tr>
</table>
```

## XML Example 1

In the XML code, note the addition of XML data tags and the nested structure of the elements.

```
<?xml version="1.0"?>
 <EMPLIST>
 <EMP>
 <EMPNO>7654</EMPNO>
 <ENAME>MARTIN</ENAME>
 <JOB>SALESMAN</JOB>
 <SAL>1250</SAL>
 </EMP>
 <EMP>
 <EMPNO>7788</EMPNO>
 <ENAME>SCOTT</ENAME>
 <JOB>ANALYST</JOB>
 <SAL>3000</SAL>
 </EMP>
 </EMPLIST>
```

## HTML Example 2

Consider the following HTML that uses tags to present data in a row of a table. Is "Java Programming" the name of a book? A university course? A job skill? You cannot be sure by looking at the data and tags on the page. Imagine a computer program trying to figure this out!

```
<HTML>
 <BODY>
 <TABLE>
 <TR>
 <TD>Java Programming</TD>
 <TD>EECS</TD>
 <TD>Paul Thompson</TD>
 <TD>Ron
Uma
Lindsay</TD>
 </TR>
 </TABLE>
 </BODY>
</HTML>
```

The analogous XML example has the same data, but the tags indicate what information the data represents, not how it should be displayed. It's clear that "Java Programming" is the Name of a Course, but it says nothing about how it should be displayed.

## XML Example 2

```
<?xml version="1.0"?>
 <Course>
 <Name>Java Programming</Name>
 <Department>EECS</Department>
 <Teacher>
 <Name>Paul Thompson</Name>
 </Teacher>
 <Student>
 <Name>Ron</Name>
 </Student>
 <Student>
 <Name>Uma</Name>
 </Student>
 <Student>
 <Name>Lindsay</Name>
 </Student>
 </Course>
```

XML and HTML both represent information:

- XML represents information *content*
- HTML represents the *presentation* of that content

## Summary of Differences Between XML and HTML

Figure 29–2 summarizes, how XML differs from HTML.

**Table 29–2 XML and HTML Differences**

XML	HTML
Represents information <i>content</i>	Represents the <i>presentation</i> of the content
Has user-defined tags	Has a fixed set of tags defined by standards.
All start tags must have end tags	Current browsers relax this requirement on tags <P>, <B>, and so on.
Attributes must be single or double quoted	Current browsers relax this requirement on tags
Empty elements are clearly indicated	Current browsers relax this requirement on tags
Element names and attributes are case sensitive	Element names and attributes are not case sensitive.

## Presenting XML Using Stylesheets

A key advantage of using XML as a datasource is that its *presentation* (such as a web page) can be separate from its *structure* and *content*.

- **Presentation.** Applied stylesheets define its *presentation*. XML data can be presented in various ways, both in appearance and organization, simply by applying different stylesheets.
- **Structure and content:** XML data defines the structure and content.

### Stylesheet Uses

Consider these ways of using stylesheets:

- A different interface can be presented to different users based on user profile, browser type, or other criteria by defining a different stylesheet for each presentation style.
- Stylesheets can be used to transform XML data into a format tailored to the specific application that receives and processes the data.

Stylesheets can be applied on the server or client side. The XSL-Transformation Processor (XSL-T Processor) transforms one XML format into XML or any other text-based format such as HTML. Oracle XML Parsers all include an XSL-T Processor.

How to apply stylesheets and use the XSL-T Processor is described in the following sections:

- [Chapter 4, "Using XSL and XSLT"](#)
- [Chapter 20, "Using XML Parser for Java"](#)
- Oracle9i Case Studies - XML Applications, under the chapter, "Customizing Discoverer4i(9i) Viewer with XSL"

## eXtensible Stylesheet Language (XSL)

eXtensible Stylesheet Language (XSL), the stylesheet language of XML is another W3C recommendation. XSL provides for stylesheets that allow you to do the following:

- Transform XML into XML or other text-based formats such as HTML
- Rearrange or filter data

- Convert XML data to XML that conforms with another Document Type Definition (DTD), an important capability for allowing different applications to share data

## Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS1), a W3C specification was originally created for use with HTML documents. With CSS you can control the following aspects of your document's appearance:

- Spacing. Element visibility, position, and size
- Colors and background
- Fonts and text

CSS2 was published by W3C in 1998 and includes the following additional features:

- System fonts and colors
- Automatic numbering
- Supports paged media
- Tables and aura stylesheets

'Cascading' here implies that you can apply several stylesheets to any one document. On a web page deploying CSS, for example, three stylesheets can apply or cascade:

1. User's preferred stylesheet takes precedence
2. Cascading stylesheet
3. Browser stylesheet

**See Also:** [Chapter 4, "Using XSL and XSLT"](#)

## Extensibility and Document Type Definitions (DTD)

Another key advantage of XML over HTML is that it leaves the specification of the tags and how they can be used to the user. You construct an XML document by creating your own tags to represent the meaning and structure of your data.

Tags may be defined by using them in an XML document or they may be formally defined in a Document Type Definition (DTD). As your data or application requirements change, you can change or add tags to reflect new data contexts or extend existing ones.

The following is a simple DTD for the previous XML example:

```
<!ELEMENT EMPLIST (EMP)*>
<!ELEMENT EMP (EMPNO, ENAME, JOB, SAL)>
<!ELEMENT EMPNO (#PCDATA)>
<!ELEMENT ENAME (#PCDATA)>
<!ELEMENT JOB (#PCDATA)>
<!ELEMENT SAL (#PCDATA)>
]>
```

---

---

**Note:** The DOCTYPE declaration is only used when the DTD is embedded in XML code.

---

---

## Well-Formed and Valid XML Documents

### Well-Formed XML Documents

An XML document that conforms to the structural and notational rules of XML is considered *well-formed*. A well-formed XML document does not have to contain or reference a DTD, but rather can implicitly define its data elements and their relationships. Well-formed XML documents must follow these rules:

- Document must start with the XML declaration, `<?xml version="1.0">`
- All elements must be contained within one root element
- All elements must be nested in a tree structure without overlapping
- All non-empty elements must have start and end tags

### Valid XML Documents

Well-formed XML documents that *also* conform to a DTD are considered *valid*. When an XML document containing or referencing a DTD is parsed, the parsing application can verify that the XML conforms to the DTD and is therefore valid, which allows the parsing application to process it with the assurance that all data elements and their content follow rules defined in the DTD.



## Why Use XML?

XML, the internet standard for information exchange is useful for the following reasons:

- ***Solves Data Interchange Problems.*** It facilitates efficient data communication where the data:
  - Is in many different formats and platforms
  - It must be sent to different platforms
  - Must appear in different formats and presentations
  - Must appear on many different end devices

In short, XML solves application *data interchange* problems. Businesses can now easily communicate with other businesses and workflow components using XML. See Chapters 2 through 20 for more information and examples of how XML solves data interchange problems.

Web-based applications can be built using XML which helps the interoperation of web, database, networking, and middleware. XML provides a structured format for data transmission.

- ***Industry-Specific Data Objects are Being Designed Using XML.*** Organizations such as OAG and XML.org are using XML to standardize data objects on a per-industry basis. This will further facilitate business-to-business data interchange.
- ***Database-Resident Data is Easily Accessed, Converted, and Stored Using XML.*** Large amounts of business data resides in relational and object-relational tables as the database provides excellent data queriability, scalability and availability. This data can be converted from XML format and stored in object-relational and pure relational database structures or generated from them back to XML for further processing.

### Other Advantages of Using XML

Other advantages of using XML include the following:

- You can make your own tags
- Many tools support XML
- XML is an Open standard

- XML parsers built according to the Open standard are interoperable parsers and avoid vendor lock-in. XML specifications are widely industry approved.
- In XML the presentation of data is separate from the data's structure and content. It is simple to customize the data's presentation. See "Presenting XML Using Stylesheets" and "Customizing Your Data Presentation".
- Universality -- XML enables the representation of data in a manner that can be self-describing and thus universally used
- Persistence -- Through the materialization of data as an XML document this data can persist while still allowing programmatic access and manipulation.
- Platform and application independence
- Scalability

## Additional XML Resources

Here are some additional resources for information about XML:

- *The Oracle XML Handbook*, Ben Chang, Mark Scardina, et.al., Oracle Press
- *Building Oracle XML Applications*, Steve Muench, O'Reilly
- *XML Bible*, Elliotte Rusty Harold, IDG Books Worldwide
- *XML Unleashed*, Morrison et al., SAMS
- *Building XML Applications*, St.Laurent and Cerami, McGraw-Hill
- *Building Web Sites with XML*, Michael Floyd, Prentice Hall PTR
- *Building Corporate Portals with XML*, Finkelstein and Aiken, McGraw-Hill
- *XML in a Nutshell*, O'Reilly
- *Learning XML - (Guide to) Creating Self-Describing Data*, Ray, O'Reilly
- <http://www.xml.com/pub/rg/46>
- <http://www.xmlmag.com/>
- <http://www.webmethods.com/>
- <http://www.infoshark.com>
- <http://www.clariant.org/>
- <http://www.xmlwriter.com/>

- [http://webdevelopersjournal.com/articles/why\\_xml.html](http://webdevelopersjournal.com/articles/why_xml.html)
- <http://www.w3schools.com/xml/>
- <http://www.w3.org/TR/REC-xml>
- <http://msdn.microsoft.com/xml/default.asp>
- <http://www.w3.org/TR> lists W3C technical reports
- <http://www.w3.org/xml> is the W3C XML activity overview page
- <http://www.xml.com> includes latest industry news about xml
- <http://www.xml-cml.org> has information about Chemical Markup Language (CML). CML documents can be viewed and edited on the Jumbo browser.
- <http://www.loc.gov/ead/> Encoded Archival Description (EAD) information developed for the US Library of Congress.
- <http://www.docuverse.com/xlf> for information about Extensible Log Format (XLF) a project to convert log files into XML log files to simplify log file administration.
- <http://www.w3.org/Math> for information about MathML which provides a way of interchanging equations between applications.
- <http://www.w3.org/AudioVideo/> for information about Synchronized Multimedia Integration Language (SMIL).
- Oracle is an official sponsor of OASIS. OASIS, <http://www.oasis-open.org>, is the world's largest independent, non-profit organization dedicated to the standardization of XML applications. It promotes participation from all industry, and brings together both competitors and overlapping standards bodies.



---

## Comparing Oracle XML Parsers and Class Generators by Language

This appendix provides a comparison of the Oracle XML Parser and Class Generators by language. The following sections are included in this appendix:

- [Comparing the Oracle XML Parsers](#)
- [Comparing the Oracle XML Class Generators](#)

## Comparing the Oracle XML Parsers

[Table B-1](#) compares the features of the Oracle XML parsers according to language.

**Figure B-1** Comparing Oracle XML Parsers

<b>Java</b>	<b>C</b>	<b>C++</b>	<b>PL/SQL</b>
<b>Parser, Version 2</b>			
Includes DOM API 2.0	Includes DOM API 1.0 and CORE 2.0	Includes DOM API, 1.0 and CORE 2.0	Includes DOM API 2.0
Includes SAX API 2.0	Includes SAX API	Includes SAX API	N/A
XSLT Processor	XSLT Processor	XSLT Processor	XSLT Processor
XML Schema Processor	XML Schema Processor	XML Schema Processor	N/A
Namespace 1.0 support	Namespace 1.0 support	Namespace 1.0 support	Namespace 1.0 support
XPath 1.0 support	XPath 1.0 support	XPath 1.0 support	XPath 1.0 support
Checks if document is well-formed	Checks if document is well-formed	Checks if document is well-formed	Checks if document is well-formed
Validating and Non-Validating Support	Validating and Non-Validating Support	Validating and Non-Validating Support	Validating and Non-Validating Support

**Figure B-1 Comparing Oracle XML Parsers (Cont.)**

<b>Java</b>	<b>C</b>	<b>C++</b>	<b>PL/SQL</b>
Character Sets (15):	Character Sets (15):	Character Sets (15):	Character Sets (12):
BIG 5	BIG 5	BIG 5	BIG 5
EBCDIC-CP.*	EBCDIC-CP.*	EBCDIC-CP.*	EBCDIC-CP.*
EUC-JP	EUC-JP	EUC-JP	EUC-JP
EUC-KR	EUC-KR	EUC-KR	EUC-KR
GB2312	GB2312	GB2312	GB2312
ISO-2022-JP	ISO-2022-JP	ISO-2022-JP	ISO-2022-JP
ISO-2022-KR	ISO-2022-KR	ISO-2022-KR	ISO-2022-KR
ISO-8859-1to -9	ISO-8859-1to -9	ISO-8859-1to -9	ISO-8859-1to -9
ISO-10646-UCS-2	ISO-10646-UCS-2	ISO-10646-UCS-2	KOI8-R
ISO-10646-UCS-4	ISO-10646-UCS-4	ISO-10646-UCS-4	Shift_JIS
KOI8-R	KOI8-R	KOI8-R	US-ASCII
Shift_JIS	Shift_JIS	Shift_JIS	UTF-8
US-ASCII	US-ASCII	US-ASCII	
UTF-8	UTF-8	UTF-8	
UTF-16	UTF-16	UTF-16	
Default Character Set:	Default Character Set:	Default Character Set:	Default Character Set:
UTF-8	UTF-8	UTF-8	UTF-8
Operating Systems:	Operating Systems:	Operating Systems:	Operating Systems:
All Oracle platforms	All Oracle platforms	All Oracle platforms	All Oracle platforms
Error recovery until fatal error	N/A	N/A	Error recovery until fatal error

## Comparing the Oracle XML Class Generators

[Table B-2](#) compares the features of the Oracle XML parsers and class generators, according to language.

**Figure B-2** Comparing Oracle XML Parsers and Class Generators

Java	C	C++	PL/SQL
<b>Class Generator</b>	N/A	-	N/A
oracle.xml.classgen oracg command line utility	N/A	xmlcg command line utility	N/A
CGDocument CGNode ClassGenerator InvalidContentException Supports DTD and XML Schema	N/A	-	N/A
Character Sets (8): EBCDIC-CP-US ISO-8859-1 ISO-10646-UCS-2 ISO-10646-UCS-4 Shift_SJIS US-ASCII UTF-8 UTF-16	N/A	Character Sets (8): EBCDIC-CP-US ISO-8859-1 ISO-10646-UCS-2 ISO-10646-UCS-4 Shift_SJIS US-ASCII UTF-8 UTF-16	N/A
Default Character Set: US-ASCII	N/A	Default Character Set: US-ASCII	N/A



---

# XDK for Java: Specifications and Cheat Sheets

This appendix describes the XDK for Java specifications and cheat sheets for each XML component for Java. The cheat sheets list the main APIs, classes, and associated methods for each XDK for Java component.

This appendix contains the following sections:

- [XML Parser for Java Cheat Sheets](#)
- [Accessing XML Parser for Java](#)
- [XDK for Java: XML Schema Processor](#)
- [XDK for Java: XML Class Generator for Java](#)
- [XDK for Java: XSQL Servlet](#)
- [XML SQL Utility for Java Cheat Sheet](#)

## XML Parser for Java Cheat Sheets

[Table C-1](#) and [Table C-2](#) list XML Parser for Java top level classes with a brief description of each. The following tables summarize other XML Parser for Java classes:

- [Table C-3, "XML Parser for Java: DTD\(\) Class Methods"](#)
- [Table C-4, "XML Parser for Java: ElementDecl\(\) Class"](#)
- [Table C-5, "XML Parser for Java: NodeFactory\(\) Class"](#)
- [Table C-6, "XML Parser for Java: NSName\(\) and NSResolver Classes"](#)
- [Table C-7, "XMLParser for Java: SAXAttrList\(\) Class"](#)
- [Table C-8, "XML Parser for Java: SAXParser\(\) Class"](#)
- [Table C-9, "XML Parser for Java: XMLParser\(\) Class"](#)

---

---

**Note:** Not all the XML Parser for Java methods are listed in the foregoing tables. For the detailed reference documentation see:

- *Oracle9i XML Reference*
  - <http://otn.oracle.com/tech/xml>
  - Your installed software under doc/
- 
- 

**Table C-1 XML Parser for Java: oracle.xml.parser.v2 Classes**

Class Summary	Description
<b>Interfaces</b>	-
NSName	This interface provides Namespace support for Element and Attr names
NSResolver	This interface provides support for resolving Namespaces
XMLDocumentHandler	This interface extends the org.xml.sax.DocumentHandler interface.
XMLToken	Basic interface for XMLToken
<b>Classes</b>	-
AttrDecl	Holds information about each attribute declared in an attribute list in the Document Type Definition.
<b>Package</b>	Implements the default behavior for the XMLDocumentHandler interface.
Oracle.xml.parser.v2	

**Table C-1 XML Parser for Java: oracle.xml.parser.v2 Classes(Cont.)**

Class Summary	Description
DOMParser	Implements an eXtensible Markup Language (XML) 1.0 parser according to the World Wide Web Consortium (W3C) recommendation.
DTD	Implements the DOM DocumentType interface and holds the Document Type Definition information for an XML document.
ElementDecl	Represents an element declaration in a DTD.
NodeFactory	Specifies methods to create various nodes of the DOM tree built during parsing.
oraxml	Provides a command-line interface to validate XML files: <pre>java oracle.xml.parser.v2.oraxml options* source</pre> <ul style="list-style-type: none"> <li>-h Prints this message</li> <li>-v Partial Validation mode</li> <li>-s Strict Validation Mode</li> <li>-w Show warnings</li> <li>-debug Debug mode</li> <li>-e &lt;error log&gt; A file to write errors to</li> </ul>
oraxsl	Provides a command-line interface to applying stylesheets on multiple XML documents: <pre>java oraxsl options* source? stylesheet? result?</pre> <ul style="list-style-type: none"> <li>-w Show warnings</li> <li>-e &lt;error log&gt; A file to write errors to</li> <li>-l &lt;xml file list&gt; List of files to transform</li> <li>-d &lt;directory&gt; Directory with files to transform</li> <li>-x &lt;source extension&gt; Extensions to exclude</li> <li>-i &lt;source extension&gt; Extensions to include</li> <li>-s &lt;stylesheet&gt; Stylesheet to use</li> <li>-r &lt;result extension&gt; Extension to use for results</li> <li>-o &lt;result extension&gt; Directory to place results</li> <li>-p &lt;param list&gt; List of Params</li> <li>-t &lt;# of threads&gt; Number of threads to use</li> <li>-v Verbose mode</li> </ul>
SAXAttrList	Implements the SAX AttributeList interface and also provides Namespace support.

**Table C-1 XML Parser for Java: oracle.xml.parser.v2 Classes(Cont.)**

<b>Class Summary</b>	<b>Description</b>
SAXParser	Implements an eXtensible Markup Language (XML) 1.0 SAX parser according to the World Wide Web Consortium (W3C) recommendation.
XMLAttr	Implements the DOM Attr interface and holds information on each attribute of an element.
XMLCDATA	Implements the DOM CDATASection interface.
XMLComment	Implements the DOM Comment interface.
XMLDocument	Implements the DOM Document interface, represents an entire XML document and serves the root of the Document Object Model tree.
XMLDocumentFragment	Implements the DOM DocumentFragment interface.
XMLElement	Implements the DOM Element interface.
XMLEntityReference	-
XMLNode	Implements the DOM Node interface and serves as the primary datatype for the entire Document Object Model.
XMLParser	Serves as a base class for the DOMParser and SAXParser classes.
XMLPI	Implements the DOM Processing Instruction interface.
XMLText	Implements the DOM Text interface.
XMLTokenizer	Implements an eXtensible Markup Language (XML) 1.0 parser according to the World Wide Web Consortium (W3C) recommendation.
XSLProcessor	Provides methods to transform an input XML document using a previously constructed XSLStylesheet.
XSLStylesheet	Holds XSL stylesheet information such as templates, keys, variables, and attribute sets.
Exceptions	-
XMLParseException	Indicates that a parsing exception occurred while processing an XML document
XSLException	Indicates that an exception occurred during XSL transformation

**Table C–2 XML Parser for Java: DOMParser() Methods**

Method	Description
<b>Constructor</b> DOMParser()	Creates a new parser object. Implements an eXtensible Markup Language (XML) 1.0 parser according to the World Wide Web Consortium (W3C) recommendation. to parse an XML document and build a DOM tree.
<b>Methods</b>	-
getDoctype()	Get the DTD
getDocument()	Gets the document
parseDTD(InputSource, String)	Parses the XML External DTD from given input source
parseDTD(InputStream, String)	Parses the XML External DTD from given input stream.
parseDTD(Reader, String)	Parses the XML External DTD from given input stream.
parseDTD(String, String)	Parses the XML External DTD from the URL indicated
parseDTD(URL, String)	Parses the XML External DTD document pointed to by the given URL and creates the corresponding XML document hierarchy.
setErrorStream(OutputStream)	Creates an output stream for the output of errors and warnings.
setErrorStream(OutputStream, String)	Creates an output stream for the output of errors and warnings.
setErrorStream(PrintWriter)	Creates an output stream for the output of errors and warnings.
setNodeFactory(NodeFactory)	Set the node factory.
showWarnings(boolean)	Switch to determine whether to print warnings

**Table C–3 XML Parser for Java: DTD() Class Methods**

DTD Class Members	Description
<b>Class</b> DTD()	Implements the DOM DocumentType interface and holds the Document Type Definition information for an XML document.
<b>Methods</b>	-
cloneNode(boolean)	Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes.
findElementDecl(String)	Finds an element declaration for the given tag name.
findEntity(String, boolean)	Finds a named entity in the DTD.
findNotation(String)	Retrieves the named notation from the DTD.

**Table C-3 XML Parser for Java: DTD() Class Methods(Cont.)**

<b>DTD Class Members</b>	<b>Description</b>
getChildNodes()	A <code>NodeList</code> that contains all children of this node.
getElementDecls()	A <code>NamedNodeMap</code> containing the element declarations in the DTD.
getEntities()	A <code>NamedNodeMap</code> containing the general entities, both external and internal, declared in the DTD.
getName()	Gets the name of the DTD, that is, the name immediately following the <code>DOCTYPE</code> keyword.
getNotations()	A <code>NamedNodeMap</code> containing the notations declared in the DTD.
getPublicId()	Gets the public identifier associated with the DTD, if specified.
getSystemId()	Gets the system identifier associated with the DTD, if specified.
hasChildNodes()	A convenience method to allow easy determination of whether a node has any children.
printExternalDTD(OutputStream)	Writes the contents of this document to the given output stream.
printExternalDTD(OutputStream, String)	Writes the contents of the external DTD to the given output stream.
printExternalDTD(PrintWriter)	Writes the contents of this document to the given output stream.

**Table C-4 XML Parser for Java: ElementDecl() Class**

<b>ElementDecl() Member Summary</b>	<b>Description</b>
<b>Class</b>	This class represents an element declaration in a DTD.
ElementDecl()	
<b>Fields</b>	-
ANY	Element content type - Children can be any element
ASTERISK	<code>ContentModelParseTreeNode</code> type - "*" node (has one children)
COMMA	<code>ContentModelParseTreeNode</code> type - "," node (has two children)
ELEMENT	<code>ContentModelParseTreeNode</code> type - 'leaf' node (has no children)
ELEMENTS	Element content type - Children can be elements as per Content Model
EMPTY	Element content type - No Children
MIXED	Element content type - Children can be PCDATA & elements as per Content Model

**Table C-4 XML Parser for Java: ElementDecl() Class(Cont.)**

<b>ElementDecl() Member Summary</b>	<b>Description</b>
OR	ContentModelParseTreeNode type - " " node (has two children)
PLUS	ContentModelParseTreeNode type - "+" node (has one children)
QMARK	ContentModelParseTreeNode type - "?" node (has one children)
<b>Methods</b>	-
expectedElements(Element)	Returns vector of element names that can be appended to the element.
findAttrDecl(String)	Gets an attribute declaration object or null if not found
getAttrDecls()	Gets an enumeration of attribute declarations
getContentElements()	Returns Vector of elements that can be appended to this element
getContentType()	Returns content model of element
getParseTree()	Returns the root node of Content Model Parse Tree.
validateContent(Element)	Validates the content of a element node.

**Table C-5 XML Parser for Java: NodeFactory() Class**

<b>NodeFactory() Member Summary</b>	<b>Description</b>
<b>Constructors</b>	-
NodeFactory()	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.
<b>Methods</b>	-
createAttribute(String, String)	Creates an attribute node with the specified tag, and text.
createCDATASection(String)	Creates a CDATA node with the specified text.
createComment(String)	Creates a comment node with the specified text.
createDocument()	Creates a document node.

**Table C-5 XML Parser for Java: NodeFactory() Class**

<b>NodeFactory() Member Summary</b>	<b>Description</b>
createElement(String)	Creates an Element node with the specified tag.
createProcessingInstruction(String, String)	Creates a PI node with the specified tag, and text.
createTextNode(String)	Creates a text node with the specified text.

**Table C-6 XML Parser for Java: NSName() and NSResolver Classes**

<b>Member Summary</b>	<b>Description</b>
<b>Class</b> NSName	Provides Namespace support for Element and Attribute names.
<b>Methods</b>	-
getExpandedName()	Gets the fully resolved name for this name
getLocalName()	Gets the local name for this name
getNamespace()	Gets the resolved Namespace for this name
getPrefix()	Gets the prefix for this name
getQualifiedName()	Gets the qualified name
<b>Class</b> NSResolver	Provides support for resolving Namespaces.
<b>Methods</b>	-
resolveNamespacePrefix(String)	Finds the namespace definition in scope for a given namespace prefix.



**Table C-7 XMLParser for Java: SAXAttrList() Class**

Member Summary	Description
<b>Class</b> SAXAttrList()	<p>Implements the SAX <code>AttributeList</code> interface and also provides Namespace support. Applications that require Namespace support can explicitly cast any attribute list returned by an Oracle parser class to <code>SAXAttrList</code> and use the methods described here. It also implements <code>Attributes (SAX 2.0)</code> interface</p> <p>This interface allows access to a list of attributes in three different ways:</p> <ul style="list-style-type: none"> <li>■ By attribute index;</li> <li>■ By Namespace-qualified name; or</li> <li>■ By qualified (prefixed) name.</li> </ul> <p>The list will not contain attributes that were declared <code>#IMPLIED</code> but not specified in the start tag. It will also not contain attributes used as Namespace declarations (<code>xmlns*</code>) unless the <code>http://xml.org/sax/features/namespace-prefixes</code> feature is set to true (it is false by default).</p> <p>If the <code>namespace-prefixes</code> feature (see above) is false, access by qualified name may not be available; if the <code>http://xml.org/sax/features/namespaces</code> feature is false, access by Namespace-qualified names may not be available.</p> <p>This interface replaces the now-deprecated <code>SAX1</code> interface, which does not contain Namespace support. In addition to Namespace support, it adds the <code>getIndex</code> methods (below). The order of attributes in the list is unspecified, and will vary with each implementation.</p>
<b>Methods</b>	-
<code>getExpandedName(int)</code>	Get the expanded name for an attribute in the list (by position)
<code>getLength()</code>	Return the number of attributes in this list.
<code>getLocalName(int)</code>	Get the local name for an attribute in the list (by position)
<code>getName(int)</code>	Return the name of an attribute in this list (by position).
<code>getNamespace(int)</code>	Get the resolved namespace for an attribute in the list (by position)
<code>getPrefix(int)</code>	Get the namespace prefix for an attribute in the list (by position)
<code>getQualifiedName(int)</code>	Get the qualified name for an attribute in the list (by position)

**Table C-7 XMLParser for Java: SAXAttrList() Class(Cont.)**

<b>Member Summary</b>	<b>Description</b>
<code>/getType(int index)</code>	<p>Looks up an attribute's type by index. The attribute type is one of the strings "CDATA", "ID", "IDREF", "IDREFS", "NMTOKEN", "NMTOKENS", "ENTITY", "ENTITIES", or "NOTATION" (always in upper case).</p> <p>If the parser has not read a declaration for the attribute, or if the parser does not report attribute types, then it must return the value "CDATA" as stated in the XML 1.0 Recommendation (clause 3.3.3, "Attribute-Value Normalization").</p> <p>For an enumerated attribute that is not a notation, the parser will report the type as "NMTOKEN".</p>
<code>getType(java.lang.String qName)</code>	Looks up an attribute's type by XML 1.0 qualified name.
<code>getType(java.lang.String uri, java.lang.String localName)</code>	Looks up an attribute's type by Namespace name.
<code>getType(String)</code>	Return the type of an attribute in the list (by name).
<code>getValue(int)</code>	Return the value of an attribute in the list (by position).
<code>getValue(String)</code>	Return the value of an attribute in the list (by name).

**Table C-8 XML Parser for Java: SAXParser() Class**

Member Summary	Description
<b>Constructors</b>	-
SAXParser()	<p>Creates a new parser object. 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.</p> <p>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. XMLReader interface contains two important enhancements over the old Parser interface:</p> <ul style="list-style-type: none"> <li>■ It adds a standard way to query and set features and properties; and</li> <li>■ It adds Namespace support, which is required for many higher-level XML standards.</li> </ul>
<b>Methods</b>	-
setDocumentHandler(DocumentHandler)	SAX applications can use this to register a new document event handler.
setDTDHandler(DTDHandler)	SAX applications can use this to register a new DTD event handler.
setEntityResolver(EntityResolver)	SAX applications can use this to register a new entity resolver
setErrorHandler(ErrorHandler)	SAX applications can use this to register a new error event handler.

**Table C–8 XML Parser for Java: SAXParser() Class (Cont.)**

Member Summary	Description
SAXParser()	<p>Creates a new parser object. 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.</p> <p>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. XMLReader interface contains two important enhancements over the old Parser interface:</p> <ul style="list-style-type: none"> <li>▪ It adds a standard way to query and set features and properties; and</li> <li>▪ It adds Namespace support, which is required for many higher-level XML standards.</li> </ul>
<b>Methods</b>	-
setDocumentHandler(DocumentHandler)	SAX applications can use this to register a new document event handler.

**Table C–9 XML Parser for Java: XMLParser() Class**

Member Summary	Description
<b>Class</b>	
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 can not be instantiated (applications can use the DOM or SAX parser depending on their requirements).
<b>Methods</b>	-
getReleaseVersion()	Returns the release version of the Oracle XML Parser
getValidationMode()	Returns the validation mode
parse(InputSource)	Parses the XML from given input source
parse(InputStream)	Parses the XML from given input stream.
parse(Reader)	Parses the XML from given input stream.

**Table C–9 XML Parser for Java: XMLParser() Class**

Member Summary	Description
parse(String)	Parses the XML from the URL indicated
parse(URL)	Parses the XML document pointed to by the given URL and creates the corresponding XML document hierarchy.
setBaseURL(URL)	Sets the base URL for loading external entities and DTDs.
setDoctype(DTD)	Sets the DTD
setLocale(Locale)	Applications can use this to set the locale for error reporting.
setPreserveWhitespace(boolean)	Sets the white space preserving mode
setValidationMode(boolean)	Sets the validation mode

## Accessing XML Parser for Java

Oracle XML Parsers are provided with Oracle Enterprise and Standard editions from release 8.1.6 and higher. If you do not have these editions you can download the XML Parsers from: <http://otn.oracle.com/tech/xml/>

## Installing XML Parser for Java, Version 2

These sections describe how to install the Windows NT and UNIX versions of the XML Parser for Java, Version 2.

**XML Parser for Java, Version 2: Windows NT Installation** To install the Oracle XML Parser for Java (v2) on Windows NT follow these steps:

1. Install JDK-1.1.x. or above and either unzip or WinZip executable.
2. Download the Oracle XML Parser in ZIP format.
3. Unzip xmlparser.zip into a directory. For example:  

```
C:\[your directory]> unzip xmlparser.zip
```
4. The result should be the following files and directories:
  - \* license.html — copy of license agreement
  - \* readme.html — release and installation notes
  - \* doc\ — directory for documents

- \* lib\ — directory for parser class files
- \* sample\ — sample code files

**XML Parser for Java, Version 2: UNIX Installation** To install the XML Parser for Java (v2) in UNIX follow these steps:

1. Install JDK-1.1.x or above and GNU `gzip`.
2. Download the Oracle XML Parser in `.tar.gz` format.
3. Extract the distribution package into a directory. For example:  

```
#gzip -dc xmlparser.tar.gz | tar xvf -
```
4. The result should be the following files and directories:
  - \* license.html — copy of license agreement
  - \* readme.html — release and installation notes
  - \* doc/ — directory for documents
  - \* lib/ — directory for parser class files
  - \* sample/ — sample code files

### Sample Code

See [Chapter 20, "Using XML Parser for Java"](#), for sample code and suggestions on how to use the XML Parsers.

## XML Parser for Java, Version 2 Specifications

The Oracle XML Parser for Java, Version 2 specifications follow:

- New high performance architecture
- Integrated support for W3C XSLT 1.0 Recommendation
- Supports validation and non-validation modes
- Built-in Error Recovery until fatal error
- Integrated Document Object Model (DOM) Level 1.0 and 2.0 API
- Integrated SAX 1.0 and 2.0 API
- Supports W3C Recommendation for XML Namespaces

## Requirements

Operating Systems: Any with Java 1.1.x support

JAVA: JDK 1.1.x. or above.

The contents of both the Windows and UNIX versions are identical. They are simply archived differently for operating system compatibility and your convenience.

## Online Documentation

Documentation for Oracle XML Parser for Java is located in the doc/ directory in your install area.

## Release Specific Notes

The readme.html file in the root directory of the archive contains release specific information including bug fixes, API additions, and so on.

Oracle XML Parser is an early adopter release and is written in Java. It will check if an XML document is well-formed and, optionally, if it is valid. The parser will construct a Java object tree which can be accessed. It also contains an integrated XSLT processor for transforming XML documents.

## Standards Conformance

The parser conforms to the following W3C Recommendations:

- Extensible Markup Language (XML) 1.0  
<http://www.w3.org/TR/1998/REC-xml-19980210>
- Namespaces in XML at  
<http://www.w3.org/TR/REC-xml-names/>
- Document Object Model Level 1 1.0  
<http://www.w3.org/TR/REC-DOM-Level-1/>
- Document Object Model Level 2  
<http://www.w3.org/TR/DOM-Level-2-Core/>
- XML Path Language (XPath) 1.0  
<http://www.w3.org/TR/1999/REC-xpath-19991116>
- XML Transformations (XSLT) 1.0  
<http://www.w3.org/TR/1999/REC-xslt-19991116>

The parser also conforms to the following W3C *Proposed Recommendations*:

- XML Schema Part 1: Structures  
<http://www.w3.org/TR/xmlschema-1>
- XML Schema Part 2: Datatypes  
<http://www.w3.org/TR/xmlschema-2>

In addition, the parser implements the following interfaces defined by the XML development community:

- Simple API for XML (SAX) 1.0 and 2.0 at  
<http://www.megginson.com/SAX/index.html>

## Supported Character Set Encodings

The XML Parser for Java currently supports the following encodings:

- BIG 5
- EBCDIC-CP-\*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1to -9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift\_JIS
- US-ASCII
- UTF-8
- UTF-16

**Default:** UTF-8 is the default encoding if none is specified. Any other ASCII or EBCDIC based encodings that are supported by the JDK may be used. However,



they must be specified in the format required by the JDK instead of as official character set names defined by IANA.

### Error Recovery

The parser also provides error recovery. It will recover from most errors and continue processing until a fatal error is encountered.

## Oracle XML Parser V1 and V2

Version 2 of the XML Parser for Java, besides incorporating an XSLT processor, has been re-architected from version 1. This has resulted in a number of changes to the class names especially those that support Namespaces. The following summarizes changes you have to take into account when converting code from v1 to v2.

---

---

**Note:** This summary is based upon XML Parser versions v1.0.1.4 as v1 and v2.0.0.0 as v2.

---

---

## NEW CLASS STRUCTURE

oracle.xml.parser package has been renamed to oracle.xml.parser.v2.

The following are new interfaces:

- NSName
- XMLDocumentHandler

The following interfaces have been removed:

- NSAttr
- NSAttributeList
- NSDocumentHandler
- NSElement

The following are new classes in v2:

- DOMParser
- DefaultXMLDocumentHandler
- SAXAttrList
- SAXParser

- XSLProcessor
- XSLStylesheet
- XSLException

[Table C-3](#) lists the XDK for Java classes that have been reorganized.

**Table C-10 XML Parser for Java: Classes Reorganization and Changes**

Version 1	Version 2
<b>Class Reorganization</b>	■
XMLParser	<ul style="list-style-type: none"> <li>■ XMLParser (includes methods applicable to DOM and SAX access),</li> <li>■ DOMParser (includes methods specific to DOM access),</li> <li>■ SAXParser (includes methods specific to SAX access)</li> </ul>
NSDocumentHandler	XMLDocumentHandler
NSAttr	XMLAttr (supports Namespace, NSAttr interface has been removed)
NSAttributeList	SAXAttrList (includes methods in NSAttributeList and org.xml.sax.AttributeList)
NSElement	XMLElement (supports Namespace, NSElement interface removed)
<b>PUBLIC CLASS / VARIABLE / CONSTRUCTOR / METHOD CHANGES</b>	-
AttrDecl	-
getName()	<eliminated> Use XMLNode.getNodeName
getPresence()	getAttrPresence()
getType()	getAttrType()
getValues()	getEnumerationValues()
<b>ElementDecl</b>	-
ASTERISK	Reserved for future implementation.
COMMA	Reserved for future implementation.
ELEMENT	Reserved for future implementation.

**Table C-10 XML Parser for Java: Classes Reorganization and Changes**

<b>Version 1</b>	<b>Version 2</b>
OR	Reserved for future implementation.
PLUS	Reserved for future implementation.
QMARK	Reserved for future implementation.
getParseTree()	Reserved for future implementation.
XMLAttr	New constructor - XMLAttr(String, String, String, String) New methods - cloneNode(), getPrefix()
XMLElement	New constructor - XMLElement(String, String, String) New methods: <ul style="list-style-type: none"> <li>▪ checkNamespace(String, String)</li> <li>▪ getElementsByTagName(String, String),</li> <li>▪ resolveNamespacePrefix(String)</li> </ul>
XMLNode	New method - transformNode()
XMLText	New method - getNodeValue()

## XDK for Java: XML Schema Processor

**Table C–12** summarizes the XML Schema Processor for Java’s classes, constructors, and methods.

**See Also:**

- [Chapter 21, "Using XML Schema Processor for Java"](#)
- The readme.txt file in your installed software’s doc/ directory. This software can also be downloaded from <http://otn.oracle.com/tech/xml>

**Table C–11 XML Schema Processor for Java: Classes, Constructors, Methods**

Members	Description
Class XMLSchema	Sets top-level XMLSchema document declarations & definitions plus schema location and schema target namespace. XMLSchema objects are created by XSDBuilder as a result of processing XMLSchema documents. They are used by XSDParser for instance XML documents validation and by XSDBuilder as imported schemas.
<b>Constructors</b> XMLSchema() XMLSchema(int)	
<b>Class</b> 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 info (min/maxOccurs).
<b>Methods</b> build(InputStream,URL)	Builds an XMLSchema object
build(Reader, URL)	Builds an XMLSchema object
build(String)	Builds an XMLSchema object
build(String, String)	Builds an XMLSchema object
build(String, URL)	Builds an XMLSchema object
build(URL)	Builds an XMLSchema object

**Table C–11 XML Schema Processor for Java: Classes, Constructors, Methods(Cont.)**

Members	Description
build(XMLDocument,URL)	Builds XMLSchema from XML document
getObject()	Returns the schema object.
setError(XMLError)	Sets XMLError object.
setLocale(Locale)	Sets locale for error reporting.
XSDException	Indicates that an exception occurred during XMLSchema validation.
Methods getMessage()	Override getMessage, in order to construct error message from error id, and error params
getMessage(XMLError)	Get localized message based on the XMLError sent as parameter

## XDK for Java: XML Class Generator for Java

Oracle XML Class Generator for Java requires Oracle XML Parser for Java. The XML Document, printed by the generated classes, confirms to the W3C recommendation for Extensible Markup Language (XML) 1.0. Oracle XML Class Generator can optionally generate validating Java source files. It also optionally generates Javadoc comments in the source files.

Oracle XML Class Generator supports the following encodings for printing the XMLDocument:

UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, US-ASCII, EBCDIC-CP-US, ISO-8859-1, and Shift\_SJIS.

ASCII is the default encoding if none is specified. Any other ASCII or EBCDIC based encodings that are supported by the JDK can be used.

## Installing XML Class Generator for Java

Installing the Oracle XML Java Class Generator, is described in the following sections.

### XML Class Generator for Java: Windows NT Installation

To install Oracle XML Class Generators on Windows NT, follow these steps:

1. Install JDK-1.1.x. or above and either unzip or run the WinZip executable.
2. Download the Oracle XML Class Generator for Java in ZIP format from

[http://otn.oracle.com/tech/xml/xdk\\_java/content.html](http://otn.oracle.com/tech/xml/xdk_java/content.html)

This Class Generator for Java uses 63KB. Select the following:

- Software, from the XML top menu >
  - Oracle XML Class Generator for Windows NT
3. Unzip `xmlclassgenV1_0_0.zip` into a directory. For example:  
`C:\[your directory]>unzip xmlclassgenV1_0_0.zip`
  4. The result should be the following files and directories:
    - `license.html` — copy of license agreement
    - `readme.html` — release and installation notes
    - `doc\` — directory for documents
    - `lib\` — directory for classgen class files
    - `sample\` — sample code files

## XML Class Generator for Java: UNIX Installation

To install Oracle XML Class Generator for Java in UNIX, follow these steps:

1. Install JDK-1.1.x or above and GNU gzip.
2. Download the Oracle XML Class Generator for Java in `.tar.gz` format from [http://otn.oracle.com/tech/xml/xdk\\_java/content.htm](http://otn.oracle.com/tech/xml/xdk_java/content.htm)  
This Class Generator for Java uses 63Kb. Select the following:
  - Software, from the XML top menu >
  - Oracle XML Class Generator for UNIX
3. Extract the distribution package into a directory. For example:  
`#gzip -dc xmlclassgenV1_0_0.tar.gz | tar xvf -`
4. The result should be the following files and directories:
  - `license.html` — copy of license agreement
  - `readme.html` — release and installation notes
  - `doc/` — directory for documents
  - `lib/` — directory for classgen class files

- sample/ — sample code files

## XML Class Generator for Java Cheat Sheet

[Table C-12](#) lists the main XML Class Generator for Java APIs and top level classes with a brief description of each. [Table C-13](#) lists the XML Class Generator for Java methods.

**Table C-12 XML Class Generator for Java: APIs and Classes**

Classes	Description
<b>Classes</b>	-
CGDocument	Serves as the base document class for the Class Generated generated classes. Constructor for the Root element of the DTD. Parameters: <code>doctype</code> - Name of the root Element of the DTD, <code>dtd</code> - The DTD used to generate the classes
CGNode	Serves as the base class for nodes generated by the Class Generated
DTDClassGenerator	Used by the Class Generator to generate classes against a DTD
SchemaClassGenerator	Used by the Class Generator to generate classes against a Schema
CGXSDElement	This class is the base class for the classes generated by Schema Class generator. The classes corresponding to the top level elements in a schema extends this class. This class contains the code for initialization and building the schema from the schema file which is required for validation purpose. Note: Since the validation is not supported, the static block for reading the schema file is not used. It should be possible to read schema file from anywhere in the scope of the CLASSPATH. It is required to read the schema file using <code>getResource</code> .
<b>Exceptions</b>	-
InvalidContentException	Definition of InvalidContentException thrown by <code>dtdcompiler</code> classes

**Table C-13 XML Class Generator for Java: Methods**

Methods	Description
<b>Class</b>	Constructor for the Root element of the DTD. public abstract class CGDocument extends CGNode. Serves as the base document class for the DTD compiler generated classes
CGDocument(String, DTD)	
<b>Methods</b>	Prints the constructed XML Document
print(OutputStream)	
print(OutputStream, String)	Prints the constructed XML Document

**Table C-13 XML Class Generator for Java: Methods(Cont.)**

<b>Methods</b>	<b>Description</b>
public abstract class CGNode	extends Object. Serves as the base class for nodes generated by the XML Class Generator.
<b>Class</b>	
generate(DTD, String)	Traverses the DTD with element doctype as root and generates Java classes
<b>Methods</b>	
setGenerateComments(boolean)	Switch to determine whether to generate java doc comments. Default - TRUE
setJavaPackage(String)	Sets the package for the classes generated. Default - No package
setOutputDirectory(String)	Sets the output directory. Default - current directory
setSerializationMode(boolean)	Switch to determine if the DTD should be saved as a serialized object or as text file.
setValidationMode(boolean)	Switch to determine whether the classes generated should validate the XML Document being constructed.Default - TRUE
<b>Class</b>	
CGNode(String)	Constructor for the Elements of the DOM Tree
<b>Methods</b>	
addCDATASection(string)	Adds CDATA Section to the Element.
addData(String)	Adds PCDATA to the Element
addNode(CGNode)	Adds a node as a child to the element
getCGDocument()	Gets the base document (root Element)
getDTDNode()	Gets the static DTD from the base document
setAttribute(String, String)	Sets the value of the Attribute
setDocument(CGDocument)	Sets the base document (root Element)
storeID(String, String)	Store this value for an ID identifier, so that we can later verify IDREF values
storeIDREF(String, String)	Store this value for an IDREF identifier, so that we can later verify, if an corresponding ID was defined.
validateContent()	Checks if the content of the element is valid as per the Content Model specified in DTD
validEntity(String)	Checks if the ENTITY identifier is valid
validID(String)	Checks if the ID identifier is valid



**Table C–13 XML Class Generator for Java: Methods(Cont.)**

Methods	Description
validNMTOKEN(String)	Checks if the NMTOKEN identifier is valid
<b>Class</b> CGXSDElement	This class serves as the base class for the all the generated classes corresponding to the XML Schema generated by Schema Class Generator
<b>Methods</b>	-
addAttribute(String, String)	Add the attribute of a given node to the hashtable. Parameters: <code>attName</code> - the attribute name, <code>attValue</code> - the attribute value
addElement(Object)	Add the elements of a given element node to the vector corresponding to the elements. Parameters: <code>elem</code> - the object which needs to be added
getAttributes()	public java.util.Hashtable getAttributes(). Return the attributes. Returns: attributes the hashtable containing attribute name and value
getChildElements()	public java.util.Vector getChildElements(). Get the vector having all the local elements. Returns: <code>elemChild</code> vector
getNodeValue()	public java.lang.String getNodeValue(). Return the node value
getType()	public java.lang.Object getType(). Return the type
print(XMLOutputStream)	public void print(oracle.xml.parser.v2.XMLOutputStream out). Print an element node. Parameters: <code>out</code> - the stream where the output is printed
printAttributes(XMLOutputStream, String)	public void printAttributes(oracle.xml.parser.v2.XMLOutputStream out, java.lang.String name). Print an attribute node. Parameters: <code>out</code> - the stream where the output is printed, <code>name</code> - the attribute name
setNodeValue(String)	protected void setNodeValue(java.lang.String value). Set the node value of an element. Parameters: <code>value</code> - the node value

## oracg Command Line Utility

`oracg` invokes the DTD or Schema Class Generator for Java to generate classes based on DTD and Schema respectively, depending on the input arguments given. [Table C–14](#) lists the `oracg` arguments.

**Table C–14 oracg Command Line Utility**

oracg Argument	Description
-h	Prints the help message text
-d <dtd file>	DTD file (.dtd file)
-s <schema file>	Schema file (.xsd file)

**Table C-14** *oracg Command Line Utility*

<b>oracg Argument</b>	<b>Description</b>
- o <Output dirname>	Output directory
- c	Comment option
- p <package name/s>	The package names corresponding to namespace

## XDK for Java: XSQL Servlet

### Downloading and Installing XSQL Servlet

#### Downloading XSQL Servlet from OTN

You can download XSQL Servlet distribution from:

[http://otn.oracle.com/tech/xml/xsql\\_servlet](http://otn.oracle.com/tech/xml/xsql_servlet)

1. Click on the 'Software' icon at the top of the page:
2. Log in with your OTN username and password (registration is free if you do not already have an account).
3. Selecting whether you want the NT or Unix download (both contain the same files)
4. Acknowledge the licensing agreement and download survey
5. Clicking on `xsqlservlet_v1.0.2.0.tar.gz` or `xsqlservlet_v1.0.2.0.zip`

#### Extracting the Files in the Distribution

To extract the contents of XSQL Servlet distribution, do the following:

1. Choose a directory under which you would like the `.\xsql` directory and subdirectories to go, for example, `C:\`
2. Change directory to `C:\`, then extract the XSQL downloaded archive file there. For example:

UNIX:

```
tar xvfz xsqlservlet_v1.0.2.0.tar.gz
```

Windows NT:

```
pkzip25 -extract -directories xsqlservlet_v1.0.2.0.zip
```

using the pkzip25 command-line tool or the WinZip visual archive extraction tool.

## Windows NT: Starting the Web-to-go Server

XSQL Servlet comes bundled with the Oracle Web-to-go server that is pre-configured to use XSQL Pages. The Web-to-go web server is a single-user server, supporting the Servlet 2.1 API, used for mobile application deployment and for development. This is a great way to try XSQL Pages out on your Windows machine before delving into the details of configuring another Servlet Engine to run XSQL Pages.

---



---

**Note:** The Web-to-go Web server is part of Oracle's development and deployment platform for mobile applications. For more information on Web-to-go, see <http://www.oracle.com/mobile>.

---



---

Windows NT users can get started quickly with XSQL Pages by following these steps:

1. Running the xsql-wtg.bat script in the .\xsql directory.
2. Browsing the URL <http://localhost:7070/xsql/index.html>

If you get an error starting this script, edit the xsql-wtg.bat file to properly set the two environment variables JAVA and XSQL\_HOME to appropriate values for your machine.

```
REM -----
REM Set the 'JAVA' variable equal to the full path
REM of your Java executable.
REM -----
set JAVA=J:\java1.2\jre\bin\java.exe
set XSQL_HOME=C:\xsql
REM -----
REM Set the 'XSQL_HOME' variable equal to the full
REM path of where you install the XSQL Servlet
REM distribution.
REM -----
```

Then, repeat the two steps above.

If you get an error connecting to the database when you try the demos, you'll need to go on to the next section, then try the steps above again after setting up your database connection information correctly in the XSQLConfig.xml file.

## Setting Up the Database Connection Definitions for Your Environment

The demos are set up to use the SCOTT schema on a database on your local machine (that is, the machine where the web server is running). If you are running a local database and have a SCOTT account whose password is TIGER, then you are all set. Otherwise, you need to edit the `.\xsql\lib\XSQLConfig.xml` file to correspond to your appropriate values for username, password, dburl, and driver values for the connection named "demo".

```
<?xml version="1.0" ?>
 <XSQLConfig>
 :
 <connectiondefs>
 <connection name="demo">
 <username>scott</username>
 <password>tiger</password>
 <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
 <driver>oracle.jdbc.driver.OracleDriver</driver>
 </connection>
 <connection name="lite">
 <username>system</username>
 <password>manager</password>
 <dburl>jdbc:Polite:Polite</dburl>
 <driver>oracle.lite.poljdbc.POLJDBCdriver</driver>
 </connection>
 </connectiondefs>
 :
 </XSQLConfig>
```

## UNIX: Setting Up Your Servlet Engine to Run XSQL Pages

UNIX users and any user wanting to install the XSQL Servlet on other web servers should continue with the instructions below depending on the web server you're trying to use. In every case, there are 3 basic steps:

1. Include the list of XSQL Java archives as well as the directory where XSQLConfig.xml resides (by default `./xsql/lib`) in the server CLASSPATH.

---

---

**Note:** For convenience, the `xsqlservlet_v1.0.2.0.tar.gz` and `xsqlservlet_v1.0.2.0.zip` distributions include the `.jar` files for the Oracle XML Parser for Java (V2), the Oracle XML SQL Utilities for Java, and the 8.1.6 JDBC driver in the `.\lib` subdirectory, along with Oracle XSQL Pages' own `.jar` archive.

---

---

2. Map the `.xsql` file extension to the `oracle.xml.xsql.XSQLServlet` servlet class
3. Map a virtual directory `/xsql` to the directory where you extracted the XSQL files (to access the on-line help and demos)

## XSQL Servlet Specifications

The following lists the XSQL servlet specifications:

- Produce dynamic XML documents based on one or more SQL queries
- Optionally transforms the resulting XML document in the server or client using XSLT
- Supports W3C XML 1.0 Recommendation
- Supports Document Object Model (DOM) Level 1.0 and 2.0 API
- Support the W3C XSLT 1.0 Recommendation
- Supports W3C Recommendation for XML Namespaces

## Character Set Support

XSQL Servlet supports the following character set encodings:

- BIG
- EBCDIC-CP-\*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1to -9

- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift\_JIS
- US-ASCII
- UTF-8
- UTF-16

## XDK for Java: XSQL Servlet Cheat Sheets

[Table C-15](#) and [Table C-16](#) list XSQL Servlet APIs and top level classes with a brief description of each.

**Table C-15 XSQL Servlet Classes**

<b>Class Summary</b>	<b>Description</b>
<b>Interfaces</b>	-
XSQLActionHandler	Interface that must be implemented by all XSQL Action Element Handlers
XSQLPageRequest	Interface representing a request for an XSQL Page
<b>Classes</b>	-
XSQLActionHandlerImpl	Base Implementation of XSQLActionHandler that can be extended to create your own custom handlers.
XSQLCommandLine	Command-line Utility to process XSQL Pages.
XSQLPageRequestImpl	Base implementation of the XSQLPageRequest interface that can be used to derive new kinds of page request implementations.
XSQLParserHelper	Common XML Parsing Routines
XSQLRequest	Programmatically process a request for an XSQL Page.
XSQLServlet	Servlet to enable HTTP GET-ing of and POST-ing to XSQL Pages
XSQLServletPageRequest	Implementation of XSQLPageRequest for Servlet-based XSQL Page requests.
XSQLStylesheetProcessor	XSLT Stylesheet Processing Engine

**Table C-16 XSQLPageRequest Interface Methods**

Methods	Description
createNestedRequest(URL, Dictionary)	Returns an instance of a nested Request
getConnectionName()	Returns the name of the connection being used for this request May be null if no connection set/in-use.
getErrorWriter()	Returns a PrintWriter to print out errors processing this request
getJDBCConnection()	Gets the JDBC connection being used for this request (can be null)
getPageEncoding()	Returns encoding of source XSQL Page associated with this request
getParameter(String)	Returns the value of the requested parameter
getPostedDocument()	Returns the content of Posted XML for this request as an XML Document
getRequestParamsAsXMLDocument()	Returns the content of a Request parameters as an XML Document
getRequestType()	Returns a string identifying the type of page request being made.
getSourceDocumentURI()	Returns a String representation of the requested document's URI
getStylesheetParameter(String)	Gets a stylesheet parameter by name
getStylesheetParameters()	Gets an enumeration of stylesheet parameter names
getStylesheetURI()	Returns the URI of the stylesheet to be used to process the result.
getUserAgent()	Returns a String identifier of the requesting program
getWriter()	Returns a PrintWriter used for writing out the results of a page request
getXSQLConnection()	Gets the XSQLConnection Object being used for this request Might be null.
isIncludedRequest()	Returns true if this request is being included in another.
isOracleDriver()	Returns true if the current connection uses the Oracle JDBC Driver
printedErrorHeader()	Returns the state of whether an Error Header has been printed
requestProcessed()	Allows Page Request to Perform end-of-request processing
setConnectionName(String)	Sets the connection name to use for this request
setContentType(String)	Sets the content type of the resulting page
setIncludingRequest(XSQLPageRequest)	Sets the Including Page Request object for this request.
setPageEncoding(String)	Sets encoding of source XSQL page associated with this request.
setPageParam(String, String)	Sets a dynamic page parameter value.
setPostedDocument(Document)	Allows programmatic setting of the Posted Document

**Table C-16 XSQLPageRequest Interface Methods (Cont.)**

<b>Methods</b>	<b>Description</b>
setPrintedErrorHandler(boolean)	Sets whether an Error Header has been printed
setStylesheetParameter(String, String)	Sets the value of a parameter to be passed to the associated stylesheet
setStylesheetURI(String)	Sets the URI of the stylesheet to be used to process the result.
translateURL(String)	Returns a string representing an absolute URL resolved relative to the base URI for this request.
useConnectionPooling()	Returns true if connection pooling is desired for this request
useHTMLErrors()	Returns true if HTML-formatted error messages are desired for this request

## XML SQL Utility for Java Cheat Sheet

See [Appendix H, "XML SQL Utility \(XSU\) Specifications and Cheat Sheets"](#).



---

# XDK for Java Beans: Specifications and Cheat Sheets

This Appendix describes the XDK for Java Bean specifications and cheat sheets.

This appendix contains the following section:

- [XDK for Javabeans: Transviewer Bean Cheat Sheet](#)
- [DOMBuilder Bean Cheat Sheet](#)
- [XSLTransformer Bean Cheat Sheet](#)
- [XMLTreeView Bean Cheat Sheet](#)
- [XMLTransformPanel Cheat Sheet](#)
- [DBViewer Bean Cheat Sheet](#)
- [XMLSourceView Bean Cheat Sheet](#)
- [DBAccess Bean Cheat Sheet](#)

## XDK for Javabeans: Transviewer Bean Cheat Sheet

The following tables list the primary classes and methods of the Transviewer Beans in XDK for Java Beans:

- [Table D-1, "DOMBuilder Bean Classes and Methods"](#)
- [Table D-2, "XSLTransformer Bean Classes and Methods"](#)
- [Table D-3, "XMLTreeView Bean Class and Methods"](#)
- [Table D-4, "XMLTransformPanel Bean Classes and Methods"](#)
- [Table D-5, "DBViewer Bean Class and Methods"](#)
- [Table D-6, "XMLSourceView Bean Classes and Methods"](#)
- [Table D-7, "DBAccess Classes and Methods"](#)

### DOMBuilder Bean Cheat Sheet

[Table D-1](#) lists the primary classes and methods in DOMBuilder Bean.

**Table D-1** *DOMBuilder Bean Classes and Methods*

<b>Class Summary</b>	<b>Description</b>
<b>Interfaces</b>	
DOMBuilderErrorListener	-
DOMBuilderListener	This interface must be implemented in order to receive notifications when error is found during parsing.
DOMBuilderListener	This interface must be implemented in order to receive notifications about events during the asynchronous parsing.
<b>Classes</b>	
DOMBuilder	-
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.
DOMBuilderBeanInfo	-

**Table D-1 DOMBuilder Bean Classes and Methods(Cont.)**

Class Summary	Description
DOMBuilderErrorEvent	This class defines the error event which is sent when parse exception occurs.
DOMBuilderEvent	The event object that DOMBuilder uses to notify all registered listeners about parse events.
ResourceManager	Simple semaphore that maintains access to fixed number of logical resources.

## XSLTransformer Bean Cheat Sheet

[Table D-2](#) lists the primary XSLTransformer Bean classes and methods.

**Table D-2 XSLTransformer Bean Classes and Methods**

Member Summary	Description
<b>Class</b>	
<b>XSLTransformer</b>	Applies XSL transformation in a background thread.
<b>Constructors</b>	
XSLTransformer()	XSLTransformer constructor
XSLTransformer(int)	XSLTransformer constructor
<b>Methods</b>	
addXSLTransformerErrorListener(XSLTransformerErrorListene r)	Adds an error event listener
addXSLTransformerListener(XSLTransformerListener)	Adds a listener
	Returns the unique XSLTransformer id
getResult()	Returns the document fragment for the resulting document.
processXSL(XSLStylesheet, InputStream, URL)	Initiate XSL Transformation in the background.
processXSL(XSLStylesheet, Reader, URL)	Initiate XSL Transformation in the background.
processXSL(XSLStylesheet, URL, URL)	Initiate XSL Transformation in the background.
	Initiate XSL Transformation in the background.
processXSL(XSLStylesheet, XMLDocument, OutputStream)	Initiate XSL Transformation in the background.

**Table D–2 XSLTransformer Bean Classes and Methods(Cont.)**

<b>Member Summary</b>	<b>Description</b>
removeDOMTransformerErrorListener(XSLTransformerErrorListener)	Removes an error event listener
removeXSLTransformerListener(XSLTransformerListener)	Removes a listener
run()	-
	Sets the error stream used by the XSL processor
showWarnings(boolean)	Sets the showWarnings flag used by the XSL processor
<b>Class</b> XSLTransformerErrorEvent	The error event object that XSLTransformer uses to notify all registered listeners about transformation error events.
<b>Class</b> XSLTransformerEvent	The event object that XSLTransformer uses to notify all registered listeners about transformation events.
<b>Class</b> XSLTransformerErrorListener	This interface must be implemented in order to receive notifications about error events during the asynchronous transformation.
<b>Class</b> XSLTransformerListener	This interface must be implemented in order to receive notifications about events during the asynchronous transformation.

## XMLTreeView Bean Cheat Sheet

[Table D–3](#) lists the XMLTreeView Bean primary class and methods.

**Table D–3 XMLTreeView Bean Class and Methods**

<b>Class or Method</b>	<b>Description</b>
<b>Class</b> XMLTreeView()	The class constructor.
<b>Methods</b>	
getPreferredSize()	Returns the XMLTreeView preferred size.
setXMLDocument(Document)	Associates the XMLTreeViewer with a XML document.
updateUI()	Forces the XMLTreeView to update/refresh UI.

**Table D–3 XMLTreeView Bean Class and Methods(Cont.)**

Class or Method	Description
Class XMLTreeViewBeanInfo()	--
Methods getIcon(int)	--
getPropertyDescriptors()	--

## XMLTransformPanel Cheat Sheet

Table D–4 lists XMLTransformPanel Bean primary classes and methods.

**Table D–4 XMLTransformPanel Bean Classes and Methods**

Classes and Methods	Description
<b>Class</b> XMLTransformPanel	XMLTransformPanel visual bean applies XSL transformations on XML documents. Visualizes the result. Allows editing of input XML and XSL documents/files.
<b>Constructors</b> XMLTransformPanel()	public XMLTransformPanel() The class constructor. Creates an object of type XMLTransformPanel.
XMLTransformPanelBeanInfo	-
<b>Constructors</b> XMLTransformPanelBeanInfo()	public XMLTransformPanelBeanInfo()
<b>Methods</b> getIcon(int)	public java.awt.Image getIcon(int iconKind) Overrides: java.beans.SimpleBeanInfo.getIcon(int) in class java.beans.SimpleBeanInfo
getPropertyDescriptors()	public java.beans.PropertyDescriptor[] getPropertyDescriptors() Overrides: java.beans.SimpleBeanInfo.getPropertyDescriptors() in class java.beans.SimpleBeanInfo

## DBViewer Bean Cheat Sheet

Table D-5 lists the DBViewer bean primary class and methods.

**Table D-5 DBViewer Bean Class and Methods**

Class and Methods	Description
<b>Class</b> DBViewer()	Constructs a new instance. Displays database queries or any XML by applying XSL stylesheets and visualizing the resulted HTML in scrollable swing panel. This bean has tree buffers: XML, XSL and result buffer. DBViewer bean allows the calling program to load/save the buffers from various sources and to apply stylesheet transformation to the XML buffer using the stylesheet in the XSL buffer. The result can be stored in the result buffer. The XML and XSL buffers content can be shown as source or as a tree structure. The result buffer content can be rendered as HTML and also shown as source or tree structure. The XML buffer can be loaded from database query. All buffers can load and save files from CLOB tables in Oracle database and from file system as well. Therefore, the control can be also used to move files between the file system and the user schema in the database.
<b>Methods</b> getHostname()	Get database host name
getInstancename()	Get database instance name
getPassword()	Get user password
getPort()	Get database port number
getResBuffer()	Get the content of the result buffer
getResCLOBFileName()	Get result CLOB file name
getResCLOBTableName()	Get result CLOB table name
getResFileName()	Get Result file name
getUsername()	Get user name
getXmlBuffer()	Get the content of the XML buffer
getXmlCLOBFileName()	Get XML CLOB file name
getXmlCLOBTableName()	Get XML CLOB table name
getXmlFileName()	Get XML file name
getXMLStringFromSQL(String)	Get XML presentation of result set from SQL query
getXslBuffer()	Get the content of the XSL buffer
getXslCLOBFileName()	Get the XSL CLOB file name

**Table D-5 DBViewer Bean Class and Methods (Cont.)**

<b>Class and Methods</b>	<b>Description</b>
getXslCLOBTableName()	Get XSL CLOB table name
getXslFileName()	Get XSL file name
loadResBuffer(String)	Load the result buffer from file
loadResBuffer(String, String)	Load the result buffer from CLOB file
loadResBufferFromClob()	Load the result buffer from CLOB file
loadResBufferFromFile()	Load the result buffer from file
loadXmlBuffer(String)	Load the XML buffer from file
loadXmlBuffer(String, String)	Load the XML buffer from CLOB file
loadXmlBufferFromClob()	Load the XML buffer from CLOB file
loadXmlBufferFromFile()	Load the XML buffer from file
loadXMLBufferFromSQL(String)	Load the XML buffer from SQL result set
loadXslBuffer(String)	Load the XSL buffer from file
loadXslBuffer(String, String)	Load the XSL buffer from CLOB file
loadXslBufferFromClob()	Load the XSL buffer from CLOB file
loadXslBufferFromFile()	Load the XSL buffer from file
parseResBuffer()	Parse the result buffer and refresh the tree view and source view
parseXmlBuffer()	Parse the XML buffer and refresh the tree view and source view
parseXslBuffer()	Parse the XSL buffer and refresh the tree view and source view
saveResBuffer(String)	Save the result buffer to file
saveResBuffer(String, String)	Save the result buffer to CLOB file
saveResBufferToClob()	Save the result buffer to CLOB file
saveResBufferToFile()	Save the result buffer to file
saveXmlBuffer(String)	Save the XML buffer to file

**Table D–5 DBViewer Bean Class and Methods (Cont.)**

<b>Class and Methods</b>	<b>Description</b>
saveXmlBuffer(String, String)	Save the XML buffer to CLOB file
saveXmlBufferToClob()	Save the XML buffer to CLOB file
saveXmlBufferToFile()	Save the XML buffer to file
saveXslBuffer(String)	Save the XSL buffer to file
saveXslBuffer(String, String)	Save the XSL buffer to CLOB file
saveXslBufferToClob()	Save the XSL buffer to CLOB file
saveXslBufferToFile()	Save the XSL buffer to file
setHostname(String)	Set database host name
setInstancename(String)	Set database instance name
setPassword(String)	Set user password
setPort(String)	Set database port number
setResBuffer(String)	Set new text in the result buffer
setResCLOBFileName(String)	Set Result CLOB file name
setResCLOBTableName(String)	Set Result CLOB table name
setResFileName(String)	Set Result file name
setResHtmlView(boolean)	Show the result buffer as rendered HTML
setResSourceEditView(boolean)	Show the result buffer as XML source and enter edit mode
setResSourceView(boolean)	Show the result buffer as XML source
setResTreeView(boolean)	Show the result buffer as XML tree view
setUsername(String)	Set user name
setXmlBuffer(String)	Set new text in the XML buffer
setXmlCLOBFileName(String)	Set XML CLOB table name



**Table D-5 DBViewer Bean Class and Methods (Cont.)**

<b>Class and Methods</b>	<b>Description</b>
setXmlCLOBTableName(String)	Set XML CLOB table name
setXmlFileName(String)	Set XML file name
setXmlSourceEditView(boolean)	Show the XML buffer as XML source and enter edit mode
setXmlSourceView(boolean)	Show the XML buffer as XML source
setXmlTreeView(boolean)	Show the XML buffer as tree
setXslBuffer(String)	Set new text in the XSL buffer
setXslCLOBFileName(String)	Set XSL CLOB file name
setXslCLOBTableName(String)	Set XSL CLOB table name
setXslFileName(String)	Set XSL file name
setXslSourceEditView(boolean)	Show the XSL buffer as XML source and enter edit mode
setXslSourceView(boolean)	Show the XSL buffer as XML source
setXslTreeView(boolean)	Show the XSL buffer as tree
transformToDoc()	Transforms the content of the XML buffer by applying the stylesheet from the XSL buffer.
transformToRes()	Apply the stylesheet transformation from the XSL buffer to the XML in the XML buffer and stores the result into the result buffer
transformToString()	Transforms the content of the XML buffer by applying the stylesheet from the XSL buffer.

## XMLSourceView Bean Cheat Sheet

[Table D-6](#) lists the primary classes and methods of XMLSourceView Bean.

**Table D-6 XMLSourceView Bean Classes and Methods**

Class and Methods	Description
<b>Class</b> 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.
<b>Fields</b> inputDOMDocument, jScrollPane, jTextPane, xmlStyledDocument	-
<b>Constructors</b> XMLSourceView()	The class constructor. Creates an object of type XMLSourceView.
<b>Methods</b> fontGet(AttributeSet)	Extracts and returns the font from a given attributeset. Parameters: attributeset - The source Attributeset. Returns: The extracted Font.
fontSet(MutableAttributeSet, Font)	Sets the mutableattributeset font. Parameters: mutableattributeset - The mutableattributeset to update, font - The new Font for the mutableattributeset.
getAttributeNameFont()	Returns the Attribute Value font. Returns: The Font object.
getAttributeNameForeground()	Returns the Attribute Name foreground color. Returns: The Color object.
getAttributeValueFont()	Returns the Attribute Value font. Returns: The Font object.
getAttributeValueForeground()	public java.awt.Color getAttributeValueForeground() Returns the Attribute Value foreground color. Returns: The Color object.
getBackground()	public java.awt.Color getBackground() Returns the background color. Overrides: java.awt.Component.getBackground() in class java.awt.Component Returns: The Color object.
getCDATAFont()	public java.awt.Font getCDATAFont() Returns the CDATA font. Returns: The Font object.
getCDATAForeground()	public java.awt.Color getCDATAForeground() Returns the CDATA foreground color. Returns: The Color object.

**Table D-6 XMLSourceView Bean Classes and Methods(Cont.)**

<b>Class and Methods</b>	<b>Description</b>
getCommentDataFont()	public java.awt.Font getCommentDataFont() Returns the Comment Data font. Returns: The Font object.
getCommentDataForeground()	public java.awt.Color getCommentDataForeground() Returns the Comment Data foreground color. Returns: The Color object
getEditedText()	public java.lang.String getEditedText() Returns the edited text. Returns: The String object containing the edited text.
getJTextPane()	public javax.swing.JTextPane getJTextPane() Returns the viewer JTextPane component. Returns: The JTextPane object used by XMLSourceViewer
getMinimumSize()	public java.awt.Dimension getMinimumSize() Returns the XMLSourceView minimal size. Overrides: javax.swing.JComponent.getMinimumSize() in class javax.swing.JComponent Returns: The Dimension object containing the XMLSourceView minimum size.
getNodeAtOffset(int)	public org.w3c.dom.Node getNodeAtOffset(int i) Returns the XML node at a given offset. Parameters: i - The node offset. Returns: The Node object from offset i.
getPCDATAFont()	public java.awt.Font getPCDATAFont() Returns the PCDATA font. Returns: The Font object.
getPCDATAForeground()	public java.awt.Color getPCDATAForeground() Returns the PCDATA foreground color. Returns: The Color object.
getPIDataFont()	public java.awt.Font getPIDataFont() Returns the PI Data font. Returns: The Font object
getPIDataForeground()	public java.awt.Color getPIDataForeground() Returns the PI Data foreground color. Returns: The Color object.
getPINameFont()	public java.awt.Font getPINameFont() Returns the PI Name font. Returns: The Font object.
getPINameForeground()	public java.awt.Color getPINameForeground() Returns the PI Data foreground color. Returns: The Color object.
getSymbolFont()	public java.awt.Font getSymbolFont() Returns the NOTATION Symbol font. Returns: The Font object.
getSymbolForeground()	public java.awt.Color getSymbolForeground() Returns the NOTATION Symbol foreground color. Returns: The Color object.
getTagFont()	public java.awt.Font getTagFont() Returns the Tag font. Returns: The Font object.

**Table D-6 XMLSourceView Bean Classes and Methods(Cont.)**

<b>Class and Methods</b>	<b>Description</b>
getTagForeground()	public java.awt.Color getTagForeground() Returns the Tag foreground color. Returns: The Color object.
getText()	public java.lang.String getText() Returns the XML document as a String. Returns: The String object containing the XML document.
isEditable()	public boolean isEditable() Returns boolean to indicate whether this object is editable.
selectNodeAt(int)	public void selectNodeAt(int i) Moves the cursor to XML Node at offset i. Parameters: i - The node offset.
setAttributeNameFont(Font)	public void setAttributeNameFont(java.awt.Font font) Sets the Attribute Name font. Parameters: font - The new Font for Attribute Name.
setAttributeNameForeground(Color)	public void setAttributeNameForeground(java.awt.Color color) Sets the Attribute Name foreground color. Parameters: color - The new Color for Attribute Name.
setAttributeValueFont(Font)	public void setAttributeValueFont(java.awt.Font font) Sets the Attribute Value font. Parameters: font - The new Font for Attribute Value.
setAttributeValueForeground(Color)	public void setAttributeValueForeground(java.awt.Color color) Sets the Attribute Value foreground color. Parameters: color - The new Color for Attribute Value.
setBackground(Color)	public void setBackground(java.awt.Color color) Sets the background color. Overrides: javax.swing.JComponent.setBackground(java.awt.Color) in class javax.swing.JComponent Parameters: color - The new background Color.
setCDATAFont(Font)	public void setCDATAFont(java.awt.Font font) Sets the CDATA font. Parameters: font - The new Font for CDATA.
setCDATAForeground(Color)	public void setCDATAForeground(java.awt.Color color) Sets the CDATA foreground color. Parameters: color - The new Color for CDATA.
setCommentDataFont(Font)	public void setCommentDataFont(java.awt.Font font) Sets the Comment font. Parameters: font - The new Font for the XML Comments.
setCommentDataForeground(Color)	public void setCommentDataForeground(java.awt.Color color) Sets the Comment foreground color. Parameters: color - The new Color for Comment.r color)

**Table D-6 XMLSourceView Bean Classes and Methods(Cont.)**

<b>Class and Methods</b>	<b>Description</b>
setEditable(boolean)	public void setEditable(boolean edit) Sets the specified boolean to indicate whether this object should be editable. Parameters: doc - The new boolean value.
setPCDATAFont(Font)	public void setPCDATAFont(java.awt.Font font) Sets the PCDATA font. Parameters: font - The new Font for PCDATA.
setPCDATAForeground(Color)	public void setPCDATAForeground(java.awt.Color color) Sets the PCDATA foreground color. Parameters: color - The new Color for PCDATA.
setPIDataFont(Font)	public void setPIDataFont(java.awt.Font font) Sets the PI Data font. Parameters: font - The new Font for PI Data.
setPIDataForeground(Color)	public void setPIDataForeground(java.awt.Color color) Sets the PI Data foreground color. Parameters: color - The new Color for PI Data.
setPINameFont(Font)	public void setPINameFont(java.awt.Font font) Sets the PI Name font. Parameters: font - The new Font for the PI Names.
setPINameForeground(Color)	public void setPINameForeground(java.awt.Color color) Sets the PI Name foreground color. Parameters: color - The new Color for PI Name.
setSelectedNode(Node)	public void setSelectedNode(org.w3c.dom.Node node) Sets the cursor position at the selected XML node. Parameters: node - The selected node.
setSymbolFont(Font)	public void setSymbolFont(java.awt.Font font) Sets the NOTATION Symbol font. Parameters: color - The new Font for NOTATION Symbol.
setSymbolForeground(Color)	public void setSymbolForeground(java.awt.Color color) Sets the NOTATION Symbol foreground color. Parameters: color - The new Color for NOTATION Symbol.
setTagFont(Font)	public void setTagFont(java.awt.Font font) Sets the Tag font. Parameters: font - The new Font for the XML Tags.
setTagForeground(Color)	public void setTagForeground(java.awt.Color color) Sets the Tag foreground color. Parameters: color - The new Color for the XML Tags.
setXMLDocument(Document)	public void setXMLDocument(org.w3c.dom.Document document) Associates the XMLviewer with a XML document. Parameters: doc - The Document document to display. See Also: getText()

## DBAccess Bean Cheat Sheet

Table D-7 lists the DBAccess Bean primary classes and methods.

**Table D-7 DBAccess Classes and Methods**

Classes and Methods	Description
<b>Class</b>	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. This CLOB tables are automatically maintained by the transviewer bean. The CLOB tables maintained by this class can be later used by the transviewer bean. The class creates and deletes CLOB tables, list a CLOB table content and also add, replace or delete text documents in this CLOB tables.
<b>Constructors</b>	public DBAccess()
DBAccess()	
<b>Methods</b>	Create BLOB table Parameters: con -- the Connection object tablename -- the table name Returns: true if successfull
createBLOBTable(Connection, String)	
createXMLTable(Connection, String)	Create XML table Parameters: con -- the Connection object, tablename -- the table name Returns: true if successfull
deleteBLOBName(Connection, String, String)	Delete binary file from BLOB table Parameters: con -- the Connection object, tablename -- the table name, xmlname -- the file name Returns: true if successfull
deleteXMLName(Connection, String, String)	Delete file from XML table Parameters: con -- the Connection object, tablename -- the table name, xmlname -- the file name Returns: true if successfull
dropBLOBTable(Connection, String)	public boolean dropBLOBTable(java.sql.Connection con, java.lang.String tablename) Delete BLOB table Parameters: con -- the Connection object, tablename -- the table name Returns: true if successfull
dropXMLTable(Connection, String)	public boolean dropXMLTable(java.sql.Connection con, java.lang.String tablename) Delete XML table Parameters: con -- the Connection object, tablename -- the table name Returns: true if successfull

**Table D-7 DBAccess Classes and Methods(Cont.)**

<b>Classes and Methods</b>	<b>Description</b>
<code>getBLOBData(Connection, String, String)</code>	<code>public byte[] getBLOBData(java.sql.Connection con, java.lang.String tablename, java.lang.String xmlname)</code> Retrieve binary file from BLOB table Parameters: con - - the Connection object, tablename - - the table name, xmlname - - the file name Returns: file as a byte array
<code>getNameSize()</code>	<code>public int getNameSize()</code> Returns the size of the field where the filename is kept. Returns: filename size
<code>getXMLData(Connection, String, String)</code>	<code>public java.lang.String getXMLData(java.sql.Connection con, java.lang.String tablename, java.lang.String xmlname)</code> Retrieve text file from XML table Parameters: con - - the Connection object, tablename - - the table name, xmlname - - the file name Returns: file as a string
<code>getXMLNames(Connection, String)</code>	<code>public java.lang.String[] getXMLNames(java.sql.Connection con, java.lang.String tablename)</code> Returns all file names in XML table Parameters: con - - the Connection object, tablename - - the table name Returns: String array with all file names in this table
<code>getXMLTableNames(Connection, String)</code>	<code>public java.lang.String[] getXMLTableNames(java.sql.Connection con, java.lang.String tablePrefix)</code> Gets all XML tables with names starting with a given string Parameters: con - - the Connection object, tablePrefix - - table prefix string Returns: array of all XML tables that begin with tablePrefix
<code>insertBLOBData(Connection, String, String, byte[])</code>	<code>public boolean insertBLOBData(java.sql.Connection con, java.lang.String tablename, java.lang.String xmlname, byte[] xmldata)</code> Inserts binary file as a row in BLOB table Parameters: con - - the Connection object, tablename - - the table name, xmlname - - the file name, xmldata - - byte array with file data Returns: true if successful
<code>insertXMLData(Connection, String, String, String)</code>	<code>public boolean insertXMLData(java.sql.Connection con, java.lang.String tablename, java.lang.String xmlname, java.lang.String xmldata)</code> Inserts text file as a row in XML table Parameters: con - - the Connection object, tablename - - the table name, xmlname - - the file name, xmldata - - string with the file data Returns: true if successful

**Table D-7 DBAccess Classes and Methods(Cont.)**

<b>Classes and Methods</b>	<b>Description</b>
isXMLTable(Connection, String)	public boolean isXMLTable(java.sql.Connection con, java.lang.String tablename) Check if the table is XML table. Parameters: con - - the Connection object, tableName - - the table name to test Returns: true if this is XML table
replaceXMLData(Connection, String, String, String)	public boolean replaceXMLData(java.sql.Connection con, java.lang.String tablename, java.lang.String xmlname, java.lang.String xmldata) Replace text file as a row in XML table Parameters: con - - the Connection object, tablename - - the table name, xmlname - - the file name, xmldata - - string with the file data Returns: true if successfull
xmlTableExists(Connection, String)	public boolean xmlTableExists(java.sql.Connection con, java.lang.String tablename) Checks if the XML table exists Parameters: con - - the Connection object, tablename - - the table name Returns: true if the table exists

**See Also:** [Chapter 23, "Using XML Transviewer Beans"](#).



---

---

# XDK for C: Specifications and Cheat Sheets

This appendix contains the following sections:

- [XML Parser for C Specifications](#)
- [XML Parser for C Revision History](#)
- [XML Parser for C: Parser Functions](#)
- [XML Parser for C: DOM API Functions](#)
- [XML Parser for C: Namespace API Functions](#)
- [XML Parser for C: XSLT API Functions](#)
- [XML Parser for C: SAX API Functions](#)

## XML Parser for C Specifications

Oracle provides a set of XML parsers for Java, C, C++, and PL/SQL. Each of these parsers is a stand-alone XML component that parses an XML document (or a standalone DTD) so that it can be processed by an application. Library and command-line versions are provided and support the following "standards" and features:

- DOM (Document Object Model) support is provided compliant with the W3C DOM 1.0 Recommendation. These APIs permit applications to access and manipulate an XML document as a tree structure in memory. This interface is used by such applications as editors.
- SAX (Simple API for XML) support is also provided compliant with the SAX 1.0 specification. These APIs permit an application to process XML documents using an event-driven model.
- Support is also included for W3C recommendation for XML Namespaces 1.0 thereby avoiding name collisions, increasing reusability and easing application integration.
- Supports validation and non-validation modes.
- Supports W3C XML 1.0 Recommendation.
- Integrated support for W3C XSLT 1.0 Recommendation.

### Validating and Non-Validating Mode Support

The XML Parser for C can parse XML in validating or non-validating modes.

- In ***non-validating mode***, the parser verifies that the XML is well-formed and parses the data into a tree of objects that can be manipulated by the DOM API.
- In ***validating mode***, the parser verifies that the XML is well-formed and validates the XML data against the DTD (if any).

Validation involves checking whether or not the attribute names and element tags are legal, whether nested elements belong where they are, and so on.

### Example Code

See [Chapter 24, "Using XML Parser for C"](#) for example code and suggestions on how to use the XML Parser for C.

## Online Documentation

Documentation for Oracle XML Parser for C is located in the `$ORACLE_HOME/xdk/c/parser/doc` directory.

## Release Specific Notes

The `readme.html` file in the root directory of the archive contains release specific information including bug fixes, API additions, and so on.

The Oracle XML parser for C is written in C. It will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface.

## Standards Conformance

XML Parser for C conforms to the following standards:

- The W3C recommendation for Extensible Markup Language (XML) 1.0 at <http://www.w3.org/TR/1998/REC-xml-19980210>
- The W3C recommendation for Document Object Model Level 1 1.0 at <http://www.w3.org/TR/REC-DOM-Level-1/>
- The W3C proposed recommendation for Namespaces in XML at <http://www.w3.org/TR/1998/PR-xml-names-19981117>
- The Simple API for XML (SAX) 1.0 at <http://www.megginson.com/SAX/index.html>
- The W3C Recommendation for XSL Transform 1.0 at <http://www.w3.org/TR/xslt>

## Supported Character Set Encodings

XML Parser for C supports documents in the following encodings, in addition to the ones specified in Appendix A, “Character Sets”, of *Oracle9i Globalization and National Language Support Guide*:

- BIG 5
- EBCDIC-CP-\*
- EUC-JP
- EUC-KR

- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1, ISO-8859-2, ISO-8859-3, ..., ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift\_JIS
- US-ASCII
- UTF-8
- UTF-16

**Default:** The default encoding is UTF-8. It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

## XML Parser for C Revision History

[Table E-1](#) lists the XML Parser for C revision history.

**Table E-1 XML Parser for C: Revision History**

Revision	Description
XML Parser 2.0.4.0.0 (C)	<p>This is the first production V2 release. This changes in this release were mainly bug fixes.</p> <p>For the XML parser, the following bugs were fixed:</p> <ul style="list-style-type: none"> <li>■ 1352943 XMLPARSE() SOMETIMES CHOKES ON FILENAMES</li> <li>■ 1302311 PROBLEM WITH PARAMETER ENTITY PROCESSING</li> <li>■ 1323674 INCONSISTENT ERROR HANDLING IN THE C XML PARSER</li> <li>■ 1328871 LPXPRINTBUFFER UNCONDITIONALLY PREPENDS XML COMMENT TO OUTPUT</li> <li>■ 1349962 USING FREED MEMORY LOCATION CAUSES TLPXVNSA31.DIF oraxmldom.h was renamed to oradom.h</li> </ul> <hr/> <p>For the XSLT processor, the following bugs were fixed:</p> <ul style="list-style-type: none"> <li>■ 1225546 USELESS ERROR MESSAGE NEEDS DETAIL</li> <li>■ 1267616 TLPXST14.DIF: REPLACE DBL_MAX WITH SBIG_ORAMAXVAL IN LPXXP.C:LPXXPSUBSTRING()</li> <li>■ 1289228 ERROR CONTEXT REQUIRED FOR DEBUGGING: FILE NAME, LINE#, FUNCTION, ETC</li> </ul> <hr/> <ul style="list-style-type: none"> <li>■ 1289214 XSL:CHOOSE DOESN'T WORK</li> <li>■ 1298028 XPATH CONSTRUCT NOT(POSITION()=LAST()) NOT WORKING</li> <li>■ 1298193 XPATH FUNCTIONS DON'T PROVIDE IMPLICIT TYPE CONVERSION OF PARAMS</li> <li>■ 1323665 C XML PARSER CANNOT SET BASE DIRECTORY OR URI FOR STYLESHEET PARSING</li> <li>■ 1325452 SEVERE MEMORY CONSUMPTION / LEAK IN XSLPROCESS</li> <li>■ 1333693 CHAINED TRANSFORMS WITH C XSL PROCESSOR DON'T WORK: LPX-00002</li> </ul>

**Table E-1 XML Parser for C: Revision History**

Revision	Description																																																	
XML Parser 2.0.3.0.0 (C)	<p>SAX memory usage: Much smaller, and flat for any input size and multiple parses (memory leaks plugged).</p> <p>XSLT memory usage: Improved. Validation warnings: Validity Constraint (VC) errors have been changed to warnings and do not terminate parsing. For compatibility with the old behavior (halt on warnings as well as errors), a new flag XML_FLAG_STOP_ON_WARNING (or '-W' to the xml program) has been added. Performance improvements: Switch to finite automata VC structure validation yields 10% performance gain.</p> <p>HTTP support: HTTP URIs are now supported; look for FTP in the next release. For other access methods, the user may define their own callbacks with the new xmlaccess() API.</p>																																																	
Oracle XML Parser 2.0.2.0.0 (C)	<p>XSLT improvements: Various bugs fixed in the XSLT processor; error messages are improved; xsl:number, xsl:sort, xsl:namespace-alias, xsl:decimal-format, forwards-compatible processing with xsl:version, and literal result element as stylesheet are now available; the following XSLT-specific additions to the core XPath library are now available: current(), format-number(), generate-id(), and system-property().</p> <p>Bug fixes: Some problems with validation and matching of start and end tags with SAX were fixed (1227096). Also, a bug with parameter entity processing in external entities was fixed (1225219).</p>																																																	
Oracle XML Parser 2.0.1.0.0 (C)	<p>Performance improvements: Major performance improvement over the last, about two and a half times faster for UTF-8 parsing and about four times faster for ASCII parsing. Comparison timing against previous version for parsing (DOM) and validating various standalone files (SPARC Ultra 1 CPU time):</p> <table border="1"> <thead> <tr> <th>File size</th> <th>Old UTF-8</th> <th>New UTF-8</th> <th>Speedup</th> <th>Old ASCII</th> <th>New ASCII</th> <th>Speedup</th> </tr> </thead> <tbody> <tr> <td>42K</td> <td>180ms</td> <td>70ms</td> <td>2.6</td> <td>120ms</td> <td>40ms</td> <td>3.0</td> </tr> <tr> <td>134K</td> <td>510ms</td> <td>210ms</td> <td>2.4</td> <td>450ms</td> <td>100ms</td> <td>4.5</td> </tr> <tr> <td>247K</td> <td>980ms</td> <td>400ms</td> <td>2.5</td> <td>690ms</td> <td>180ms</td> <td></td> </tr> <tr> <td>3.81M</td> <td>2860ms</td> <td>1130ms</td> <td>2.5</td> <td>1820ms</td> <td>380ms</td> <td>4.82</td> </tr> <tr> <td>7M</td> <td>10550ms</td> <td>4100ms</td> <td>2.6</td> <td>7450ms</td> <td>1930ms</td> <td>3.9</td> </tr> <tr> <td>10.5M</td> <td>42250ms</td> <td>16400ms</td> <td>2.6</td> <td>29900ms</td> <td>7800ms</td> <td>3.8.</td> </tr> </tbody> </table>	File size	Old UTF-8	New UTF-8	Speedup	Old ASCII	New ASCII	Speedup	42K	180ms	70ms	2.6	120ms	40ms	3.0	134K	510ms	210ms	2.4	450ms	100ms	4.5	247K	980ms	400ms	2.5	690ms	180ms		3.81M	2860ms	1130ms	2.5	1820ms	380ms	4.82	7M	10550ms	4100ms	2.6	7450ms	1930ms	3.9	10.5M	42250ms	16400ms	2.6	29900ms	7800ms	3.8.
File size	Old UTF-8	New UTF-8	Speedup	Old ASCII	New ASCII	Speedup																																												
42K	180ms	70ms	2.6	120ms	40ms	3.0																																												
134K	510ms	210ms	2.4	450ms	100ms	4.5																																												
247K	980ms	400ms	2.5	690ms	180ms																																													
3.81M	2860ms	1130ms	2.5	1820ms	380ms	4.82																																												
7M	10550ms	4100ms	2.6	7450ms	1930ms	3.9																																												
10.5M	42250ms	16400ms	2.6	29900ms	7800ms	3.8.																																												

**Table E-1 XML Parser for C: Revision History**

Revision	Description
	<p>Conformance improvements: Stricter conformance to the XML 1.0 spec yields higher scores on standard test suites (Jim Clark, Oasis,...).</p> <p>Lists, not arrays: Internal parser data structures are now uniformly lists; arrays have been dropped. Therefore, access is now better suited to a firstChild/nextSibling style loop instead of numChildNodes/getChildNode.</p> <p>DTD parsing: A new API call <code>xmlparsedtd()</code> is added which parses an external DTD directly, without needing an enclosing document. Used mainly by the Class Generator.</p>
	<p>Error reporting: Error messages are improved and more specific, with nearly twice as many as before. Error location is now described by a stack of line number/entity pairs, showing the final location of the error and intermediate inclusions (e.g. line X of file, line Y of entity).</p> <p>NOTE: You must use the new error message file (<code>lpxus.msb</code>) provided with this release; the error message file provided with earlier releases is incompatible. See below. XSL improvements: Various bugs fixed in the XSLT processor; <code>xsl:call-template</code> is now fully supported.</p>
Oracle XML Parser 2.0.0.0.0 (C)	<p>Oracle XML v2 parser is a beta release and is written in C. The main difference from the Oracle XML v1 parser is the ability to format the XML document according to a stylesheet via an integrated an XSLT processor. The XML parser will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface.</p> <p>Supported operating systems are Solaris 2.6, Linux 2.2, HP-UX 11.0, and NT 4 / Service Pack 3 (and above). Be sure to read the licensing agreement before using this product.</p>

## XML Parser for C: Parser Functions

[Table E-2](#) lists the XML Parser for C Parser functions, a brief description, and syntax.

**Table E-2 XML Parser for C: Parser Functions**

Function	Brief Description	Syntax and Comments
xmlinit	Initialize XML parser	<code>xmlctx *xmlinit (uword *err, const oratext *encoding, void (*msghdlr)(void *msgctx, const oratext *msg, ub4 errcode), void *msgctx, const xmlsaxcb *saxcb, void *saxcbctx, const xmlmemcb *memcb, void *memcbctx, const oratext *lang);</code>
xmlclean	Clean up memory used during parse	<code>void xmlclean(xmlctx *ctx);</code> For those who want to parse multiple files but would like to free the memory used for parses before the subsequent call to <code>xmlparse()</code> or <code>xmlparsebuf()</code> .
xmlparse	Parse a file	<code>uword xmlparse(xmlctx *ctx, const oratext *filename, const oratext *encoding, ub4 flags);</code> Flag bits must be OR'd to override the default behavior of the parser. The following flag bits may be set: <ul style="list-style-type: none"> <li>■ <code>XML_FLAG_VALIDATE</code> turns validation on.</li> <li>■ <code>XML_FLAG_DISCARD_WHITESPACE</code> discards whitespace where it appears to be insignificant.</li> </ul> The default behavior is to not validate the input. The default behavior for whitespace processing is to be fully conformant to the XML 1.0 spec, i.e. all whitespace is reported back to the application but it is indicated which whitespace is ignorable.
xmlparsebuf	Parse a buffer	<code>uword xmlparsebuf(xmlctx *ctx, const oratext *buffer, size_t len, const oratext *encoding, ub4 flags);</code>
xmlterm	Shut down XML parser	<code>uword xmlterm(xmlctx *ctx);</code>
createDocument	Create a new document	<code>xmlNode* createDocument(xmlctx *ctx)</code> An XML document is always rooted in a node of type <code>DOCUMENT_NODE</code> -- this function creates that root node and sets it in the context.
isStandalone	Return document's standalone flag	<code>boolean isStandalone(xmlctx *ctx)</code> Returns the boolean value of the document's standalone flag, as specified in the <code>&lt;?xml?&gt;</code> processing instruction.



## XML Parser for C: DOM API Functions

[Table E-3](#) lists the XML Parser for C DOM API functions.

**Table E-3 XML Parser for C: DOM API Functions**

<b>Function</b>	<b>Brief Description</b>
appendChild	Append child node to current node
appendData	Append character data to end of node's current data
cloneNode	Create a new node identical to the current one
createAttribute	Create a new attribute for an element node
createCDATASection	Create a CDATA_SECTION node
createComment	Create a COMMENT node
createDocumentFragment	Create a DOCUMENT_FRAGMENT node
createElement	Create an ELEMENT node
createEntityReference	Create an ENTITY_REFERENCE node
createProcessingInstruction	Create a PROCESSING_INSTRUCTION (PI) node
createTextNode	Create a TEXT node
deleteData	Remove substring from a node's character data
getAttributeName	Return an attribute's name
getAttributeSpecified	Return value of attribute's specified flag [DOM getSpecified]
getAttributeValue	Return the value of an attribute
getAttribute	Return the value of an attribute
getAttributeIndex	Return an element's attribute given its index
getAttributeNode	Get an element's attribute node given its name [DOM getName]
getAttributes	Return array of element's attributes
getCharData	Return character data for a TEXT node [DOM getData]
getCharLength	Return length of TEXT node's character data [DOM getLength]
getChildNode	Return indexed node from array of nodes [DOM item]
getChildNodes	Return array of node's children

**Table E-3 XML Parser for C: DOM API Functions (Cont.)**

<b>Function</b>	<b>Brief Description</b>
getContentModel	Returns the content model for an element from the DTD [DOM extension]
getDocument	Return top-level DOCUMENT node [DOM extension]
getDocumentElement	Return highest-level (root) ELEMENT node
getDocType	Returns current DTD
getDocTypeEntities	Returns array of DTD's general entities
getDocTypeName	Returns name of DTD
getDocTypeNotations	Returns array of DTD's notations
getElementsByTagName	Returns list of elements with matching name
getEntityNotation	Returns an entity's NDATA [DOM getNotation]
getEntityPubID	Returns an entity's public ID [DOM getPublicId]
getEntitySysID	Returns an entity's system ID [DOM getSystemId]
getFirstChild	Returns the first child of a node
getImplementation	Returns DOM-implementation structure (if defined)
getLastChild	Returns the last child of a node
getModifier	Returns a content model node's '?', '*', or '+' modifier [DOM extension]
getNextSibling	Returns a node's next sibling
getNamedItem	Returns the named node from a list of nodes
getNodeMapLength	Returns number of entries in a NodeMap [DOM getLength]
getNodeName	Returns a node's name
getNodeTypeInfo	Returns a node's type code (enumeration)
getNodeValue	Returns a node's "value", its character data
getNotationPubID	Returns a notation's public ID [DOM getPublicId]
getNotationSysID	Returns a notation's system ID [DOM getSystemId]
getOwnerDocument	Returns the DOCUMENT node containing the given node
getPIData	Returns a processing instruction's data [DOM getData]

**Table E-3 XML Parser for C: DOM API Functions (Cont.)**

<b>Function</b>	<b>Brief Description</b>
getPITarget	Returns a processing instruction's target [DOM getTarget]
getParentNode	Returns a node's parent node
getPreviousSibling	Returns a node's "previous" sibling
getTagName	Returns a node's "tagname", same as name for now
hasAttributes	Determines if element node has attributes [DOM extension]
hasChildNodes	Determines if node has children
hasFeature	Determines if DOM implementation supports a specific feature
insertBefore	Inserts a new child node before the given reference node
insertData	Inserts new character data into a node's existing data
isStandalone	Determines if document is standalone [DOM extension]
nodeValid	Validates a node against the current DTD [DOM extension]
normalize	Normalize a node by merging adjacent TEXT nodes
numAttributes	Returns number of element node's attributes [DOM extension]
numChildNodes	Returns number of node's children [DOM extension]
removeAttribute	Removes an element's attribute given its names
removeAttributeNode	Removes an element's attribute given its pointer
removeChild	Removes a node from its parents list of children
removeNamedItem	Removes a node from a list of nodes given its name
replaceChild	Replaces one node with another
replaceData	Replaces a substring of a node's character data with another string setAttribute Sets (adds or replaces) a new attribute for an element node given the attribute's name and value setAttributeNode Sets (adds or replaces) a new attribute for an element node given a pointer to the new attribute
setNamedItem	Sets (adds or replaces) a new node in a parent's list of children
setNodeValue	Sets a node's "value" (character data)
setPIData	Sets a processing instruction's data [DOM setData]

## XML Parser for C: Namespace API Functions

[Table E-4](#) lists the XML Parser for C, Namespace functions.

**Table E-4 XML Parser for C: Namespace API Functions**

<b>Function</b>	<b>Brief Description</b>
<code>getAttrLocal(xmlattr *attrs)</code>	Returns attribute local name
<code>getAttrNamespace(xmlattr *attr)</code>	Returns attribute namespace (URI)
<code>getAttrPrefix(xmlattr *attr)</code>	Returns attribute prefix
<code>getAttrQualified_name(xmlattr *attr)</code>	Returns attribute fully qualified name
<code>getNodeLocal(xmlnode *node)</code>	Returns node local name
<code>getNodeNamespace(xmlnode *node)</code>	Returns node namespace (URI)
<code>getNodePrefix(xmlnode *node)</code>	Returns node prefix
<code>getNodeQualified_name(xmlnode *node)</code>	Returns node qualified name

## XML Parser for C: XSLT API Functions

[Table E-5](#) lists the XML Parser for C, XSLT functions.

**Table E-5 XML Parser for C: XSLT API Functions**

<b>Function</b>	<b>Brief Description</b>
<code>xslprocess()</code>	Processes XSL Stylesheet with XML document source and returns success or an error code.
<code>xslprocess(xmlctx *docctx, xmlctx *xslctx, xmlctx *resctx, xmlnode **result)</code>	

## XML Parser for C: SAX API Functions

**Table E-6** lists the XML Parser for C, SAX API functions.

**Table E-6 XML Parser for C: SAX API Functions**

<b>SAX Function</b>	<b>Brief Description</b>
characters(void *ctx, const oratext *ch, size_t len)	Receive notification of character data inside an element.
endDocument(void *ctx)	Receive notification of the end of the document.
endElement(void *ctx, const oratext *name)	Receive notification of the end of an element.
ignorableWhitespace(void *ctx, const oratext *ch, size_t len)	Receive notification of ignorable whitespace in element content.
notationDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId)	Receive notification of a notation declaration.
processingInstruction(void *ctx, const oratext *target, const oratext *data)	Receive notification of a processing instruction.
startDocument(void *ctx)	Receive notification of the beginning of the document.
startElement(void *ctx, const oratext *name, const struct xmlattrs *attrs)	Receive notification of the start of an element.
unparsedEntityDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId, const oratext *notationName)	Receive notification of an unparsed entity declaration.
<b>Non-SAX Callback Functions</b>	-
nsStartElement(void *ctx, const oratext *qname, const oratext *local, const oratext *namespace, const struct xmlattrs *attrs)	Receive notification of the start of a namespace for an element.



---

# XDK for C++: Specifications and Cheat Sheet

This appendix contains the following sections:

- [XML Parser for C++ Specifications](#)
- [XML Parser for C++ Revision History](#)
- [XML Parser for C++: XMLParser\(\) API](#)
- [XML Parser for C++: DOM API](#)
- [XML Parser for C++: XSLT API](#)
- [XML Parser for C++: SAX API](#)
- [XML C++ Class Generator Specifications](#)

## XML Parser for C++ Specifications

Oracle provides a set of XML parsers for Java, C, C++, and PL/SQL. Each of these parsers is a stand-alone XML component that parses an XML document (or a standalone DTD) so that it can be processed by an application. Library and command-line versions are provided supporting the following standards and features:

- DOM (Document Object Model) support is provided compliant with the W3C DOM 1.0 Recommendation. These APIs permit applications to access and manipulate an XML document as a tree structure in memory. This interface is used by such applications as editors.
- SAX (Simple API for XML) support is also provided compliant with the SAX 1.0 specification. These APIs permit an application to process XML documents using an event-driven model.
- Support is also included for W3C recommendation for XML Namespaces 1.0 thereby avoiding name collisions, increasing reusability and easing application integration.
- Supports validation and non-validation modes
- Supports W3C XML 1.0 Recommendation
- Integrated support for W3C XSLT 1.0 Recommendation

### Validating and Non-Validating Mode Support

The XML Parser for C++ can parse XML in validating or non-validating modes.

- In ***non-validating mode***, the parser verifies that the XML is well-formed and parses the data into a tree of objects that can be manipulated by the DOM API.
- In ***validating mode***, the parser verifies that the XML is well-formed and validates the XML data against the DTD (if any).

Validation involves checking whether or not the attribute names and element tags are legal, whether nested elements belong where they are, and so on.

### Example Code

See [Chapter 26, "Using XML Parser for C++"](#) for example code and suggestions on how to use the XML Parser for C++.



## Online Documentation

Documentation for Oracle XML Parser for C++ is located in the \$ORACLE\_HOME/xdk/cpp/parser/doc directory.

## Release Specific Notes

The readme.html file in the root directory of the archive contains release specific information including bug fixes, API additions, and so on.

The Oracle XML parser for C++ is written in C with C++ wrappers. It will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface.

## Standards Conformance

XML Parser for C++ conforms to the following standards:

- The W3C recommendation for Extensible Markup Language (XML) 1.0 at <http://www.w3.org/TR/1998/REC-xml-19980210>
- The W3C recommendation for Document Object Model Level 1 1.0 at <http://www.w3.org/TR/REC-DOM-Level-1/>
- The W3C proposed recommendation for Namespaces in XML at <http://www.w3.org/TR/1998/PR-xml-names-19981117>
- The Simple API for XML (SAX) 1.0 at <http://www.megginson.com/SAX/index.html>
- The W3C Recommendation for XSL Transform 1.0 at <http://www.w3.org/TR/xslt>

## Supported Character Set Encodings

XML Parser for C++ supports documents in the following encodings, in addition to the ones specified in Appendix A, “Character Sets”, of *Oracle9i Globalization and National Language Support Guide*:

- BIG 5
- EBCDIC-CP-\*
- EUC-JP
- EUC-KR

- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1, ISO-8859-2, ISO-8859-3, ..., ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift\_JIS
- US-ASCII
- UTF-8
- UTF-16

**Default:** The default encoding is UTF-8. It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

## XML Parser for C++ Revision History

[Table F-1](#) lists the XML Parser for C++ revision history.

**Table F-1 XML Parser for C++: Revision History**

Revision	Description
Oracle XML Parser 2.0.4.0.0 (C++)	<p>First production v2 release. Changes are mainly bug fixes.</p> <p>For XML parser, the following bugs were fixed:</p> <ul style="list-style-type: none"> <li>■ 1352943 XMLPARSE() SOMETIMES CHOKES ON FILENAMES</li> <li>■ 1302311 PROBLEM WITH PARAMETER ENTITY PROCESSING</li> <li>■ 1323674 INCONSISTENT ERROR HANDLING IN THE C XML PARSER</li> <li>■ 1328871 LPXPRINTBUFFER UNCONDITIONALLY PREPENDS XML COMMENT TO OUTPUT</li> <li>■ 1349962 USING FREED MEMORY LOCATION CAUSES TLPXVNSA31.DIF oraxmlDOM.h was renamed to oradom.h</li> </ul>
-	<p>For the XSLT processor, the following bugs were fixed:</p> <ul style="list-style-type: none"> <li>■ 1225546 USELESS ERROR MESSAGE NEEDS DETAIL</li> <li>■ 1267616 TLPXST14.DIF: REPLACE DBL_MAX WITH SBIG_ORAMAXVAL IN LPXXP.C:LPXXPSUBSTRING()</li> <li>■ 1289228 ERROR CONTEXT REQUIRED FOR DEBUGGING: FILE NAME, LINE#, FUNCTION, ETC</li> </ul>
-	<ul style="list-style-type: none"> <li>■ 1289214 XSL:CHOOSE DOESN'T WORK</li> <li>■ 1298028 XPATH CONSTRUCT NOT(POSITION()=LAST()) NOT WORKING</li> <li>■ 1298193 XPATH FUNCTIONS DON'T PROVIDE IMPLICIT TYPE CONVERSION OF PARAMS</li> <li>■ 1323665 C XML PARSER CANNOT SET BASE DIRECTORY OR URI FOR STYLESHEET PARSING</li> <li>■ 1325452 SEVERE MEMORY CONSUMPTION / LEAK IN XSLPROCESS</li> <li>■ 1333693 CHAINED TRANSFORMS WITH C XSL PROCESSOR DON'T WORK: LPX-00002</li> </ul>

**Table F-1 XML Parser for C++: Revision History (Cont.)**

Revision	Description																																																	
Oracle XML Parser 2.0.3.0.0 (C++)	<p>SAX memory usage: Smaller, and flat for any input size and multiple parses (memory leaks plugged).</p> <p>XSLT memory usage: Improved.</p> <p>Validation warnings: Validity Constraint (VC) errors have been changed to warnings and do not terminate parsing. For compatibility with the old behavior (halt on warnings as well as errors), a new flag <code>XML_FLAG_STOP_ON_WARNING</code> (or '-W' to the xml program) has been added.</p> <p>Performance improvements: Switch to finite automata VC structure validation yields 10% performance gain.</p> <p>HTTP support: HTTP URIs are now supported; look for FTP in the next release. For other access methods, the user may define their own callbacks with the new <code>xmlaccess()</code> API.</p>																																																	
Oracle XML Parser 2.0.2.0.0 (C++)	<p>XSLT improvements: Various bugs fixed in the XSLT processor; error messages are improved; <code>xsl:number</code>, <code>xsl:sort</code>, <code>xsl:namespace-alias</code>, <code>xsl:decimal-format</code>, forwards-compatible processing with <code>xsl:version</code>, and literal result element as stylesheet are now available; the following XSLT-specific additions to the core XPath library are now available: <code>current()</code>, <code>format-number()</code>, <code>generate-id()</code>, and <code>system-property()</code>.</p> <p>XML parser bug fixes: Some problems with validation and matching of start and end tags with SAX were fixed. Also, a bug with parameter entity processing in external entities was fixed.</p>																																																	
Oracle XML Parser 2.0.1.0.0 (C++)	<p>Performance improvements: Major performance improvement over the last, about two and a half times faster for UTF-8 parsing and about four times faster for ASCII parsing. Comparison timing against previous version for parsing (DOM) and validating various standalone files (SPARC Ultra 1 CPU time):</p> <table border="1"> <thead> <tr> <th>File size</th> <th>Old UTF-8</th> <th>New UTF-8</th> <th>Speedup</th> <th>Old ASCII</th> <th>New ASCII</th> <th>Speedup</th> </tr> </thead> <tbody> <tr> <td>42K</td> <td>180ms</td> <td>70ms</td> <td>2.6</td> <td>120ms</td> <td>40ms</td> <td>3.0</td> </tr> <tr> <td>134K</td> <td>510ms</td> <td>210ms</td> <td>2.4</td> <td>450ms</td> <td>100ms</td> <td>4.5</td> </tr> <tr> <td>247K</td> <td>980ms</td> <td>400ms</td> <td>2.5</td> <td>690ms</td> <td>180ms</td> <td>3.8</td> </tr> <tr> <td>1M</td> <td>2860ms</td> <td>1130ms</td> <td>2.5</td> <td>1820ms</td> <td>380ms</td> <td>4.8</td> </tr> <tr> <td>2.7M</td> <td>10550ms</td> <td>4100ms</td> <td>2.6</td> <td>7450ms</td> <td>1930ms</td> <td>3.9</td> </tr> <tr> <td>10.5M</td> <td>42250ms</td> <td>16400ms</td> <td>2.6</td> <td>29900ms</td> <td>7800ms</td> <td>3.8</td> </tr> </tbody> </table> <p>Conformance improvements: Stricter conformance to the XML 1.0 spec yields higher scores on standard test suites (Jim Clark, Oasis, etc).</p>	File size	Old UTF-8	New UTF-8	Speedup	Old ASCII	New ASCII	Speedup	42K	180ms	70ms	2.6	120ms	40ms	3.0	134K	510ms	210ms	2.4	450ms	100ms	4.5	247K	980ms	400ms	2.5	690ms	180ms	3.8	1M	2860ms	1130ms	2.5	1820ms	380ms	4.8	2.7M	10550ms	4100ms	2.6	7450ms	1930ms	3.9	10.5M	42250ms	16400ms	2.6	29900ms	7800ms	3.8
File size	Old UTF-8	New UTF-8	Speedup	Old ASCII	New ASCII	Speedup																																												
42K	180ms	70ms	2.6	120ms	40ms	3.0																																												
134K	510ms	210ms	2.4	450ms	100ms	4.5																																												
247K	980ms	400ms	2.5	690ms	180ms	3.8																																												
1M	2860ms	1130ms	2.5	1820ms	380ms	4.8																																												
2.7M	10550ms	4100ms	2.6	7450ms	1930ms	3.9																																												
10.5M	42250ms	16400ms	2.6	29900ms	7800ms	3.8																																												

**Table F-1 XML Parser for C++: Revision History (Cont.)**

Revision	Description																																																
-	<p>Lists, not arrays: Internal parser data structures are now uniformly lists; arrays have been dropped. Therefore, access is now better suited to a firstChild/nextSibling style loop instead of numChildNodes/getChildNode. DTD parsing: A new API call xmlparsedtd() is added which parses an external DTD directly, without needing an enclosing document. Used mainly by the Class Generator.</p>																																																
-	<p>Error reporting: Error messages are improved and more specific, with nearly twice as many as before. Error location is now described by a stack of line number/entity pairs, showing the final location of the error and intermediate inclusions (e.g. line X of file, line Y of entity).</p> <p>NOTE: You must use the new error message file (lpxus.msb) provided with this release; the error message file provided with earlier releases is incompatible. See below.</p> <p>XSL improvements: Various bugs fixed in the XSLT processor; xsl:call-template is now fully supported.</p>																																																
Oracle XML Parser 2.0.1.0.0 (C++)	<p>Performance improvements: Major performance improvement over the last, about two and a half times faster for UTF-8 parsing and about four times faster for ASCII parsing. Comparison timing against previous version for parsing (DOM) and validating various standalone files (SPARC Ultra 1 CPU time):</p> <table border="1"> <thead> <tr> <th>File size</th> <th>Old</th> <th>New</th> <th>Speedup</th> </tr> </thead> <tbody> <tr> <td>42K</td> <td>180ms</td> <td>70ms</td> <td>2.61</td> </tr> <tr> <td>120ms</td> <td>40ms</td> <td>3.01</td> <td>34K</td> </tr> <tr> <td>510ms</td> <td>210ms</td> <td>2.44</td> <td>50ms</td> </tr> <tr> <td>100ms</td> <td>4.52</td> <td>47K</td> <td>980ms</td> </tr> <tr> <td>400ms</td> <td>2.56</td> <td>90ms</td> <td>180ms</td> </tr> <tr> <td>3.81M</td> <td>2860ms</td> <td>1130ms</td> <td>2.51</td> </tr> <tr> <td>820ms</td> <td>380ms</td> <td>4.82</td> <td>7M</td> </tr> <tr> <td>10550ms</td> <td>4100ms</td> <td>2.67</td> <td>450ms</td> </tr> <tr> <td>1930ms</td> <td>3.91</td> <td>0.5M</td> <td>42250ms</td> </tr> <tr> <td>16400ms</td> <td>2.62</td> <td>9900ms</td> <td>7800ms</td> </tr> <tr> <td>3.8</td> <td></td> <td></td> <td></td> </tr> </tbody> </table>	File size	Old	New	Speedup	42K	180ms	70ms	2.61	120ms	40ms	3.01	34K	510ms	210ms	2.44	50ms	100ms	4.52	47K	980ms	400ms	2.56	90ms	180ms	3.81M	2860ms	1130ms	2.51	820ms	380ms	4.82	7M	10550ms	4100ms	2.67	450ms	1930ms	3.91	0.5M	42250ms	16400ms	2.62	9900ms	7800ms	3.8			
File size	Old	New	Speedup																																														
42K	180ms	70ms	2.61																																														
120ms	40ms	3.01	34K																																														
510ms	210ms	2.44	50ms																																														
100ms	4.52	47K	980ms																																														
400ms	2.56	90ms	180ms																																														
3.81M	2860ms	1130ms	2.51																																														
820ms	380ms	4.82	7M																																														
10550ms	4100ms	2.67	450ms																																														
1930ms	3.91	0.5M	42250ms																																														
16400ms	2.62	9900ms	7800ms																																														
3.8																																																	
-	<p>Conformance improvements: Stricter conformance to the XML 1.0 spec yields higher scores on standard test suites (Jim Clark, Oasis, etc).</p> <p>Lists, not arrays: Internal parser data structures are now uniformly lists; arrays have been dropped. Therefore, access is now better suited to a firstChild/nextSibling style loop instead of numChildNodes/item.</p>																																																
-	<p>DTD parsing: A new method XMLParser::xmlparseDTD() is added which parses an external DTD directly, without needing an enclosing document. Used mainly by the Class Generator.</p>																																																

**Table F-1 XML Parser for C++: Revision History (Cont.)**

<b>Revision</b>	<b>Description</b>
-	Error reporting: Error messages are improved and more specific, with nearly twice as many as before. Error location is now described by a stack of line number/entity pairs, showing the final location of the error and intermediate inclusions (e.g. line X of file, line Y of entity).
-	NOTE: Use the new error message file (lpxus.msb) provided with this release; the error message file provided with earlier releases is incompatible. See below.  XSL improvements: Various bugs fixed in the XSLT processor; xsl:call-template is now fully supported.
Oracle XML Parser 2.0.0.0.0 (C++)	The Oracle XML v2 parser is a beta release and is written in C, with a C++ wrapper. The main difference from the Oracle XML v1 parser is the ability to format the XML document according to a stylesheet via an integrated an XSLT processor. The XML parser will check if an XML document is well-formed, and optionally validate it against a DTD. The parser will construct an object tree which can be accessed via a DOM interface or operate serially via a SAX interface.

## XML Parser for C++: XMLParser() API

**Table F-2** lists the main XML Parser for C++, class XMLParser() methods with a brief description of each. XMLParser() class contains top-level methods that do the following:

- Invoke the parser
- Return high-level information about a document

**Table F-2 XML Parser for C++: XMLParser() Class**

XMLParser() Method	Description
xmlinit	Initialize XML parser uword xmlinit(oratext *encoding, void (*msghdlr)(void *msgctx, oratext *msg, ub4 errcode), void *msgctx, lpxsaxcb *saxcb, void *saxcbctx, oratext *lang)
xmlterm	Terminate XML parser
xmlparse	Parse a document from a file
xmlparseBuffer	Parse a document from a buffer
getContent	Returns the content model for an element
getModifier	Returns the modifier ('?', '*' or '+') for a content-model node
getDocument	Returns the root node of a parsed document
getDocumentElement	Returns the root element (node) of a parsed document
getDocType	eturns the document type string
isStandalone	Returns the value of the standalone flag. Returns TRUE if the document is specified as standalone on the <?xml?> line, FALSE otherwise.

## XML Parser for C++: DOM API

[Table F-3](#) lists the XML Parser for C++ DOM API methods a brief description of each.

**Table F-3 XML Parser for C++: DOM API Classes (SubClasses)**

Class (Subclass)	Methods	Description
<b>Attr (Node)</b>		
This class contains methods for accessing the name and value of a single document node attribute.		
	getName	Return name of attribute
	getValue	Return "value" (definition) of attribute
	getSpecified	Return attribute's "specified" flag value
	setValue	Set an attribute's value
<b>CDATASection (Text)</b>		
This class implements the CDATA node type, a subclass of Text. There are no methods.		
<b>CharacterData (Node)</b>		
This class contains methods for accessing and modifying the data associated with text nodes.		
	appendData	Append a string to this node's data
	deleteData	Remove a substring from this node's data
	getData	Get data (value) of a text node
	getLength	Return length of a text node's data
	insertData	Insert a string into this node's data
	replaceData	Replace a substring in this node's data
	substringData	Fetch a substring of this node's data



**Table F-3 XML Parser for C++: DOM API Classes (SubClasses) (Cont.)**

Class (Subclass)	Methods	Description
<b>Comment- (CharacterData)</b>	-	-
This class implements the COMMENT node type, a subclass of CharacterData. There are no methods.		
<b>Document (Node)</b>	-	-
This class contains methods for creating and retrieving nodes.		
	createAttribute	Create an ATTRIBUTE node
	createCDATASection	Create a CDATA node
	createComment	Create a COMMENT node
	createDocumentFragment	Create a DOCUMENT_FRAGMENT node
	createElement	Create an ELEMENT node
	createEntityReference	Create an ENTITY_REFERENCE node
	createProcessingInstruction	Create a PROCESSING_INSTRUCTION node
	createTextNode	Create a TEXT node
	getElementsByTagName	Select nodes based on tag name
	getImplementation	Return DTD for document
<b>DocumentFragment (Node)</b>	-	-
This class implements the DOCUMENT_FRAGMENT node type, a subclass of Node.		
<b>DocumentType (Node)</b>	-	-
This class contains methods for accessing information about the Document Type Definition (DTD) of a document.		
	getName	Return name of DTD

**Table F-3 XML Parser for C++: DOM API Classes (SubClasses) (Cont.)**

<b>Class (Subclass)</b>	<b>Methods</b>	<b>Description</b>
	getEntities	Return NamedNodeMap of DTD's (general) entities
	getNotations	Return NamedNodeMap of DTD's notations
<b>DOMImplementation</b>	-	-
This class contains methods relating to the specific DOM implementation supported by the parser.		
	hasFeature	Detect if the named feature is supported
	Element (Node	This class contains methods pertaining to element nodes.
	getTagName	Return the node's tag name
	getAttribute	Select an attribute given its name
	setAttribute	Create a new attribute given its name and value
	removeAttribute	Remove an attribute given its name
	getAttributeNode	Remove an attribute given its name
	setAttributeNode	Add a new attribute node
	removeAttributeNode	Remove an attribute node
	getElementsByTagName	Return a list of element nodes with the given tag name
	normalize	"Normalize" an element (merge adjacent text nodes)
<b>Entity (Node)</b>	-	-
This class implements the ENTITY node type, a subclass of Node.		
	getNotation	NameReturn entity's NDATA (notation name)
	getPublicId	Return entity's public ID
	getSystemId	Return entity's system ID

**Table F-3 XML Parser for C++: DOM API Classes (SubClasses) (Cont.)**

Class (Subclass)	Methods	Description
<b>EntityReference (Node)</b> This class implements the ENTITY_REFERENCE node type, a subclass of Node.	-	-
<b>NamedNodeMap</b> This class contains methods for accessing the number of nodes in a node map and fetching individual nodes.	-	-
	item	Return nth node in map
	getLength	Return number of nodes in map
	getNamedItem	Select a node by name
	setNamedItem	Set a node into the map
	removeNamedItem	Remove the named node from map
<b>Node</b> This class contains methods for details about a document node	-	-
	appendChild	Append a new child to the end of the current node's list of children
	cloneNode	Clone an existing node and optionally all its children
	getAttributes	Return structure contains all defined node attributes
	getChildNode	Return specific indexed child of given node
	getChildNodes	Return structure contains all child nodes of given node
	getFirstChild	Return first child of given node
	getLastChild	Return last child of given node
	getLocalName	Returns the local name of the node
	getNamespaceURI	Return a node's namespace
	getNextSibling	Return a node's next sibling

**Table F-3 XML Parser for C++: DOM API Classes (SubClasses) (Cont.)**

<b>Class (Subclass)</b>	<b>Methods</b>	<b>Description</b>
	getName	Return name of node
	getType	Return numeric type-code of node
	getValue	Return "value" (data) of node
	getOwnerDocument	Return document node which contains a node
	getParentNode	Return parent node of given node
	getPrefix	Returns the namespace prefix for the node
	getPreviousSibling	Returns the previous sibling of the current node
	getQualifiedName	Return namespace qualified node of given node
	hasAttributes	Determine if node has any defined attributes
	hasChildNodes	Determine if node has children
	insertBefore	Insert new child node into a node's list of children
	numChildNodes	Return count of number of child nodes of given node
	removeChild	Remove a node from the current node's list of children
	replaceChild	Replace a child node with another
	setValue	Sets a node's value (data)
<b>NodeList</b>	-	-
This class contains methods for extracting nodes from a NodeList		
	item	Return nth node in list
	getLength	Return number of nodes in list
<b>Notation (Node)</b>	-	-
This class implements the NOTATION node type, a subclass of Node.		
	getData	Return notation's data
	getTarget	Return notation's target
	setData	Set notation's data

**Table F-3 XML Parser for C++: DOM API Classes (SubClasses) (Cont.)**

Class (Subclass)	Methods	Description
<b>ProcessingInstruction (Node)</b>	-	-
This class implements the PROCESSING_INSTRUCTION node type, a subclass of Node.		
	getData	Return the PI's data
	getTarget	Return the PI's target
	setData	Set the PI's data
<b>Text (CharacterData)</b>	-	-
This class contains methods for accessing and modifying the data associated with text nodes (subclasses CharacterData).		
	splitText	Get data (value) of a text node

## XML Parser for C++: XSLT API

XSLT is a language for transforming XML documents into other XML documents. It is designed for use as part of XSL, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.

XSLT is also designed to be used independently of XSL. However, XSLT is not intended as a completely general-purpose XML transformation language. Rather it is designed primarily for the kinds of transformation that are needed when XSLT is used as part of XSL.

A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely different from the

structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added.

## Stylesheets

A transformation expressed in XSLT is called a stylesheet. This is because, in the case when XSLT is transforming into the XSL formatting vocabulary, the transformation functions as a stylesheet.

A stylesheet contains a set of template rules. A template rule has two parts:

- A pattern which is matched against nodes in the source tree
- A template which can be instantiated to form part of the result tree. This allows a stylesheet to be applicable to a wide class of documents that have similar source tree structures.

## How Stylesheet Templates are Processed

A template is instantiated for a particular source element to create part of the result tree. A template can contain elements that specify literal result element structure. A template can also contain elements from the XSLT namespace that are instructions for creating result tree fragments. When a template is instantiated, each instruction is executed and replaced by the result tree fragment that it creates.

Instructions can select and process descendant source elements. Processing a descendant element creates a result tree fragment by finding the applicable template rule and instantiating its template. Note that elements are only processed when they have been selected by the execution of an instruction. The result tree is constructed by finding the template rule for the root node and instantiating its template.

A software module called an XSL processor reads XML documents and transforms them into other XML documents with different styles.

XML Parser for C++ implementation of the XSL processor follows the XSL Transformations standard (version 1.0, November 16, 1999) and includes the required behavior of an XSL processor as specified in the XSLT specification.

[Table F-4](#) lists the XSLProcessor class methods and syntax summary.

**Table F-4 XML Parser for C++: XSLProcessor Class**

Class	Method
XSLProcessor	xslprocess()
This class contains top-level methods for invoking the XSL processor.	Processes an XSL stylesheet with an XML document source. Syntax: uword xslprocess(XMLParser *docctx, XMLParser *xslctx, XMLParser *resctx, Node **result); where: 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

## XML Parser for C++: SAX API

The SAX API is based on callbacks. Instead of the entire document being parsed and turned into a data structure which may be referenced (by the DOM interface), the SAX interface is serial. As the document is processed, appropriate SAX user callback functions are invoked. Each callback function returns an error code, zero meaning success, any non-zero value meaning failure. If a non-zero code is returned, document processing is stopped.

To use SAX, an `xmlsaxcb` structure is initialized with function pointers and passed to the `xmlinit()` call. A pointer to a user-defined context structure may also be included; that context pointer will be passed to each SAX function.

This SAX functionality is identical to the XML Parser for C version.

[Table F-5](#) lists the XML Parser for C++, SAX API functions.

**Table F-5 XML Parser for C++: SAX API Functions**

SAX Function	Brief Description
<code>characters(void *ctx, const oratext *ch, size_t len)</code>	Receive notification of character data inside an element.
<code>endDocument(void *ctx)</code>	Receive notification of the end of the document.
<code>endElement(void *ctx, const oratext *name)</code>	Receive notification of the end of an element.

**Table F-5 XML Parser for C++: SAX API Functions**

<b>SAX Function</b>	<b>Brief Description</b>
ignorableWhitespace(void *ctx, const oratext *ch, size_t len)	Receive notification of ignorable whitespace in element content.
notationDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId)	Receive notification of a notation declaration.
processingInstruction(void *ctx, const oratext *target, const oratext *data)	Receive notification of a processing instruction.
startDocument(void *ctx)	Receive notification of the beginning of the document.
startElement(void *ctx, const oratext *name, const struct xmlattrs *attrs)	Receive notification of the start of an element.
unparsedEntityDecl(void *ctx, const oratext *name, const oratext *publicId, const oratext *systemId, const oratext *notationName)	Receive notification of an unparsed entity declaration.
<b>Non-SAX Callback Functions</b>	-
nsStartElement(void *ctx, const oratext *qname, const oratext *local, const oratext *namespace, const struct xmlattrs *attrs)	Receive notification of the start of a namespace for an element.



## XML C++ Class Generator Specifications

Working in conjunction with the XML Parser for C++, the XML Class Generator generates a set of C++ source files based on an input DTD. The generated C++ source files can then be used to construct, optionally validate, and print a XML document that is compliant to the DTD specified. The Class Generator supports validation mode to assist debugging.

### Input to the XML C++ Class Generator

Input is an XML document containing a DTD. The document body itself is ignored; only the DTD is relevant, though the dummy document must conform to the DTD. The underlying XML parser only accepts file names for the document and associated external entities. In future releases, no dummy document will be required, and URIs for additional protocols will be accepted.

### Character Set Support

The following lists supported Character Set Encoding for files input to XML C++ Class Generator. These are in addition to the character sets specified in Appendix A, "Character Sets", of *Oracle9i Globalization and National Language Support Guide*.

- BIG 5
- EBCDIC-CP-\*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1, ISO-8859-2, ISO-8859-3, ..., ISO-8859-9
- ISO-10646-UCS-2
- ISO-10646-UCS-4
- KOI8-R
- Shift\_JIS
- US-ASCII
- UTF-8

- UTF-16

**Default:** The default encoding is UTF-8. It is recommended that you set the default encoding explicitly if using only single byte character sets (such as US-ASCII or any of the ISO-8859 character sets) for performance up to 25% faster than with multibyte character sets, such as UTF-8.

## Output to XML C++ Class Generator

XML Parser for C++ output is a pair of C++ source files, .cpp and .h, named after the DTD. Constructors are provided for each class (element) that allow an object to be created in two different ways: initially empty, then adding the children or data after the initial creation, or created with the initial full set of children or initial data. A method is provided for #PCDATA (and Mixed) elements to set the data and, when appropriate, set an element's attributes.

## Standards Conformance

XML C++ Class Generator conforms to the following "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

## Directory Structure

The XML C++ Class Generator has the following file and directory structure:

```
license.html licensing agreement
bin/ Standalone Class Generator "xmlcg"
doc/ API documentation
include/ Header files
lib/ XML and support libraries
msg/ Error message files (including cause/action information in the
 .msg)
sample/ Example usage
```

[Table F-6](#) lists the libraries included with XML C++ Class Generator.

**Table F-6 XML C++ Class Generator Libraries**

<b>XML C++ Class Generator Library</b>	<b>Description</b>
libxml8.a	XML Parser/XSL Processor
libxmlg8.a	XML Class Generator
libxmlc8.a	Compatibility library needed to link with Oracle 8.1.5
libcore8.a	CORE functions
libnls8.a	National Language Support



---

# XDK for PL/SQL: Specifications and Cheat Sheets

This Appendix describes Oracle XDK for PL/SQL specifications and includes syntax cheat sheets. It contains the following sections:

- [XML Parser for PL/SQL](#)
- [XML Parser for PL/SQL Specifications](#)
- [XML Parser for PL/SQL: Parser\(\) API](#)
- [XML Parser for PL/SQL: XSLT Processor API](#)
- [XML Parser for PL/SQL: W3C DOM API — Types](#)
- [XML Parser for PL/SQL: W3C DOM API — Node Methods, Node Types, and DOM Interface Types](#)

## XML Parser for PL/SQL

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.

### Oracle XML Parser Features

The XML Parser for PL/SQL parses an XML document (or a standalone DTD) so that it can be processed by an application. Library and command-line versions are provided supporting the following standards and features:

- DOM (Document Object Model) support is provided compliant with the W3C DOM 1.0 Recommendation. These APIs permit applications to access and manipulate an XML document as a tree structure in memory. This interface is used by such applications as editors.
- SAX (Simple API for XML) support is also provided compliant with the SAX 1.0 specification. These APIs permit an application to process XML documents using an event-driven model.
- Support is also included for XML Namespaces 1.0 thereby avoiding name collisions, increasing reusability and easing application integration.
- Able to run on Oracle and Oracle Application Server.
- C and C++ versions initially available for Windows, Solaris, and Linux.

Additional features include:

- Validating and non-validating operation modes
- Built-in error recovery until fatal error
- DOM extension APIs for document creation Oracle XSL-Transform Processors

Version 2 of the Oracle XML Parsers include an integrated XSL-Transformation (XSL-T) Processor for transforming XML data using XSL stylesheets. Using the XSL-T processor, you can transform XML documents from XML to XML, HTML, or virtually any other text-based format. These processors support the following standards and features:

- Compliant with the W3C XSL Transform Proposed Recommendation 1.0
- Compliant with the W3C XPath Proposed Recommendation 1.0
- Integrated into the XML Parser for improved performance and scalability
- Available with library and command-line interfaces for Java, C, C++, and PL/SQL

## Namespace Support

The Java, C, and C++ parsers also support XML Namespaces. Namespaces are a mechanism to resolve or avoid name collisions between element types (tags) or attributes in XML documents. This mechanism provides "universal" namespace element types and attribute names whose scope extends beyond the containing document. Such tags are qualified by uniform resource identifiers (URIs), such as `<oracle:EMP xmlns:oracle="http://www.oracle.com/xml"/>`. For example, namespaces can be used to identify an Oracle `<EMP>` data element as distinct from another company's definition of an `<EMP>` data element. This enables an application to more easily identify elements and attributes it is designed to process. The Java, C, and C++ parsers support namespaces by being able to recognize and parse universal element types and attribute names, as well as unqualified "local" element types and attribute names.

## Validating and Non-Validating Mode Support

The Java, C, and C++ parsers can parse XML in validating or non-validating modes. In non-validating mode, the parser verifies that the XML is well-formed and parses the data into a tree of objects that can be manipulated by the DOM API. In validating mode, the parser verifies that the XML is well-formed and validates the XML data against the DTD (if any). Validation involves checking whether or not the attribute names and element tags are legal, whether nested elements belong where they are, and so on.

## Example Code

See [Chapter 29, "Using XML Parser for PL/SQL"](#) for example code and suggestions on how to use the XML Parsers.

## IXML Parser for PL/SQL Directory Structure

The following lists the XML Parser for PL/SQL directory structure in `$ORACLE_HOME/xdk/plsql/parser`:

- Windows NT
  - license.html - copy of license agreement
  - readme.html - release and installation notes
  - doc\ - directory for parser apis.
  - lib\ - directory for parser sql and class files
  - sample\ - sample code
- UNIX
  - license.html — copy of license agreement
  - readme.html — release and installation notes
  - doc/ — directory for parser apis
  - lib/ — directory for parser sql and class files
  - sample/ — sample code files

## DOM and SAX APIs

XML APIs generally fall into two categories: event-based and tree-based. An event-based API (such as SAX) uses callbacks to report parsing events to the application. The application deals with these events through customized event handlers. Events include the start and end of elements and characters. Unlike tree-based APIs, event-based APIs usually do not build in-memory tree representations of the XML documents. Therefore, in general, SAX is useful for applications that do not need to manipulate the XML tree, such as search operations, among others. For example, the following XML document:

```
<?xml version="1.0"?>
 <EMPLIST>
 <EMP>
 <ENAME>MARTIN</ENAME>
 </EMP>
 <EMP>
 <ENAME>SCOTT</ENAME>
 </EMP>
 </EMPLIST>
```

Becomes a series of linear events:

```
start document
start element: EEMPLIST
```



```
start element: EMP
start element: ENAME
characters: MARTIN
end element: EMP
start element: EMP
start element: ENAME
characters: SCOTT
end element: EMP
end element: EMPLIST
end document
```

A tree-based API (such as DOM) builds an in-memory tree representation of the XML document. It provides classes and methods for an application to navigate and process the tree. In general, the DOM interface is most useful for structural manipulations of the XML tree, such as reordering elements, adding or deleting elements and attributes, renaming elements, and so on.

## XML Parser for PL/SQL Specifications

These are the Oracle XML Parser for PL/SQL specifications:

- Supports validation and non-validation modes
- Includes built-in error recovery until fatal error
- Supports the W3C XML 1.0 Recommendation
- Supports the W3C XSL-T Final Working Draft

This PL/SQL implementation of the XML processor (or parser) follows 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.

### XML Parser for PL/SQL: Default Behavior

The following is the default behavior for this PLSQL 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

The types and methods described in this document are made available by the PLSQL package `xmlparser`.

- Integrated Document Object Model (DOM) Level 1.0 API

### Supported Character Set Encodings

Supports documents in the following Oracle database encodings:

- BIG 5
- EBCDIC-CP-\*
- EUC-JP
- EUC-KR
- GB2312
- ISO-2022-JP
- ISO-2022-KR
- ISO-8859-1to -9
- KOI8-R
- Shift\_JIS
- US-ASCII
- UTF-8

**Default:** UTF-8 is the default encoding if none is specified. Any other ASCII or EBCDIC based encodings that are supported by the Oracle database may be used.

### Requirements

Oracle database with the Java option enabled.

### Online Documentation

Documentation for Oracle XML Parser for PL/SQL is located in the doc directory in your install area and also in *Oracle9i XML Reference*.

### Release Specific Notes

The Oracle XML parser for PL/SQL is an early adopter release and is written in PL/SQL and Java. It will check if an XML document is well-formed and, optionally, if it is valid. The parser will construct an object tree which can be accessed via PLSQL interfaces.

## Standards Conformance

The parser conforms to the following standards:

W3C recommendation for Extensible Markup Language (XML) 1.0 at <http://www.w3.org/TR/1998/REC-xml-19980210>

W3C recommendation for Document Object Model Level 1 1.0 at <http://www.w3.org/TR/REC-DOM-Level-1/>

The parser currently does not currently have SAX or Namespace support. These will be made available in a future version.

## Error Recovery

The parser also provides error recovery. It will recover from most errors and continue processing until a fatal error is encountered.

Important note: The contents of both the Windows and UNIX versions are identical. They are simply archived differently for operating system compatibility and your convenience.

# XML Parser for PL/SQL: Parser() API

Table G–1 lists the XML Parser for PL/SQL Parser() API functions.

**Table G–1** XML Parser for PL/SQL: Parser() API

Parser() Functions	Description
parse(VARCHAR2)	Parses xml stored in the given url/file and returns the built DOM Document
newParser	Returns a new parser instance
parse(Parser, VARCHAR2)	Parses xml stored in the given url/file
parseBuffer(Parser, VARCHAR2)	Parses xml stored in the given buffer
parseClob(Parser, CLOB)	Parses xml stored in the given clob
parseDTD(Parser, VARCHAR2, VARCHAR2)	Parses xml stored in the given url/file
parseDTDBuffer(Parser, VARCHAR2, VARCHAR2)	Parses xml stored in the given buffer
parseDTDClob(Parser, CLOB, VARCHAR2)	Parses xml stored in the given clob
setBaseDir(Parser, VARCHAR2)	Sets base directory used to resolve relative urls
showWarnings(Parser, BOOLEAN)	Turn warnings on or off

**Table G–1 XML Parser for PL/SQL: Parser() API**

<b>Parser() Functions</b>	<b>Description</b>
setErrorLog(Parser, VARCHAR2)	Sets errors to be sent to the specified file
setPreserveWhitespace(Parser, BOOLEAN)	Sets white space preserve mode
setValidationMode(Parser, BOOLEAN)	Sets validation mode
getValidationMode(Parser)	Gets validation mode
setDoctype(Parser, DOMDocumentType)	Sets DTD
getDoctype(Parser)	Gets DTD
getDocument(Parser)	Gets DOM document
freeParser(Parser)	Frees a Parser object

## XML Parser for PL/SQL: XSLT Processor API

Table G-2 lists the XML Parser for PL/SQL XSL-T Processor API functions.

for the following interfaces:

- Processor interface type: Processor()
- Stylesheet interface type: Stylesheet()

**Table G-2 XML Parser for PL/SQL: XSL-T Processor() API Functions**

<b>XSL-T Processor Functions</b>	<b>Description</b>
newProcessor	Returns a new processor instance
processXML(Processor, Stylesheet, DOMDocument)	Transforms input XML document using given DOMDocument and stylesheet
processXML(Processor, Stylesheet, DOMDocumentFragment)	Transforms input XML document fragment using given DOMDocumentFragment and stylesheet
showWarnings(Processor, BOOLEAN)	Turn warnings on or off
setErrorLog(Processor, VARCHAR2)	Sets errors to be sent to the specified file
newStylesheet(DOMDocument, VARCHAR2)	Returns a new stylesheet using the given DOMDocument and reference URL
newStylesheet(VARCHAR2, VARCHAR2)	Returns a new stylesheet using the given input and reference URLs
transformNode(DOMNode, Stylesheet)	Transforms a node in a DOM tree using the given stylesheet
selectNodes(DOMNode, VARCHAR2)	Selects nodes from a DOM tree which match the given pattern
selectSingleNode(DOMNode, VARCHAR2)	Selects the first node from the tree that matches the given pattern
valueOf(DOMNode, VARCHAR2)	Retrieves the value of the first node from the tree that matches the given pattern
setParam(Stylesheet, VARCHAR2, VARCHAR2)	Sets the value of a top-level stylesheet parameter
removeParam(Stylesheet, VARCHAR2)	Remove a top-level stylesheet parameter
resetParams(Stylesheet)	Resets the top-level stylesheet parameters
freeStylesheet(Stylesheet)	Free a stylesheet object
freeProcessor(Processor)	Free a processor object

## XML Parser for PL/SQL: W3C DOM API — Types

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

The XML Parser for PL/SQL W3C DOM APIs are listed on OTN at the following site: <http://otn.oracle.com/tech/xml>

**Table G–3 XML Parser for PL/SQL: W3C DOM API Types**

Types	DOMException types	DOM interface types
DOM Node types	INDEX_SIZE_ERR	DOMNode
ELEMENT_NODE	DOMSTRING_SIZE_ERR	DOMNamedNodeMap
ATTRIBUTE_NODE	HIERARCHY_REQUEST_ERR	DOMNodeList
TEXT_NODE	WRONG_DOCUMENT_ERR	DOMAttr
CDATA_SECTION_NODE	INVALID_CHARACTER_ERR	DOMCDataSection
ENTITY_REFERENCE_NODE	NO_DATA_ALLOWED_ERR	DOMCharacterData
ENTITY_NODE	NO_MODIFICATION_ALLOWED_ERR	DOMComment
PROCESSING_INSTRUCTION_NODE	NOT_FOUND_ERR	DOMDocumentFragment
COMMENT_NODE	NOT_SUPPORTED_ERR	DOMElement
DOCUMENT_NODE	INUSE_ATTRIBUTE_ERR	DOMEntity
DOCUMENT_TYPE_NODE	DOMException types	DOMEntityReference
DOCUMENT_FRAGMENT_NODE	INDEX_SIZE_ERR	DOMNotation
NOTATION_NODE	DOMSTRING_SIZE_ERR	DOMProcessingInstruction
-	-	DOMText
-	-	DOMImplementation
-	-	DOMDocumentType
-	-	DOMDocument

# XML Parser for PL/SQL: W3C DOM API — Node Methods, Node Types, and DOM Interface Types

## Node Methods

The following lists the DOM API Node methods:

- FUNCTION isNull(n DOMNode) RETURN BOOLEAN;
- FUNCTION makeAttr(n DOMNode) RETURN DOMAttr;
- FUNCTION makeCDATASection(n DOMNode) RETURN DOMCDATASection;
- FUNCTION makeCharacterData(n DOMNode) RETURN DOMCharacterData;
- FUNCTION makeComment(n DOMNode) RETURN DOMComment;
- FUNCTION makeDocumentFragment(n DOMNode) RETURN DOMDocumentFragment;
- FUNCTION makeDocumentType(n DOMNode) RETURN DOMDocumentType;
- FUNCTION makeElement(n DOMNode) RETURN DOMELEMENT;
- FUNCTION makeEntity(n DOMNode) RETURN DOMEntity;
- FUNCTION makeEntityReference(n DOMNode) RETURN DOMEntityReference;
- FUNCTION makeNotation(n DOMNode) RETURN DOMNotation;
- FUNCTION makeProcessingInstruction(n DOMNode) RETURN DOMProcessingInstruction;
- FUNCTION makeText(n DOMNode) RETURN DOMText;
- FUNCTION makeDocument(n DOMNode) RETURN DOMDocument;
- PROCEDURE writeToFile(n DOMNode, fileName VARCHAR2);
- PROCEDURE writeToBuffer(n DOMNode, buffer IN OUT VARCHAR2);
- PROCEDURE writeToClob(n DOMNode, cl IN OUT CLOB);
- PROCEDURE writeToFile(n DOMNode, fileName VARCHAR2, charset VARCHAR2);
- PROCEDURE writeToBuffer(n DOMNode, buffer IN OUT VARCHAR2, charset VARCHAR2);

- PROCEDURE writeToClob(n DOMNode, cl IN OUT CLOB, charset VARCHAR2);
- FUNCTION getNodeName(n DOMNode) RETURN VARCHAR2;
- FUNCTION getNodeValue(n DOMNode) RETURN VARCHAR2;
- PROCEDURE setNodeValue(n DOMNode, nodeValue IN VARCHAR2);
- FUNCTION getNodeType(n DOMNode) RETURN NUMBER;
- FUNCTION getParentNode(n DOMNode) RETURN DOMNode;
- FUNCTION getChildNodes(n DOMNode) RETURN DOMNodeList;
- FUNCTION getFirstChild(n DOMNode) RETURN DOMNode;
- FUNCTION getLastChild(n DOMNode) RETURN DOMNode;
- FUNCTION getPreviousSibling(n DOMNode) RETURN DOMNode;
- FUNCTION getNextSibling(n DOMNode) RETURN DOMNode;
- FUNCTION getAttributes(n DOMNode) RETURN DOMNamedNodeMap;
- FUNCTION getOwnerDocument(n DOMNode) RETURN DOMDocument;
- FUNCTION insertBefore(n DOMNode, newChild IN DOMNode, refChild IN DOMNode) RETURN DOMNode;
- FUNCTION replaceChild(n DOMNode, newChild IN DOMNode, oldChild IN DOMNode) RETURN DOMNode;
- FUNCTION removeChild(n DOMNode, oldChild IN DOMNode) RETURN DOMNode;
- FUNCTION appendChild(n DOMNode, newChild IN DOMNode) RETURN DOMNode;
- FUNCTION hasChildNodes(n DOMNode) RETURN BOOLEAN;
- FUNCTION cloneNode(n DOMNode, deep boolean) RETURN DOMNode;

## DOM Node Types

The following lists the DOM API Node types:

- DOM Node types
- ELEMENT\_NODE
- ATTRIBUTE\_NODE



- TEXT\_NODE
- CDATA\_SECTION\_NODE
- ENTITY\_REFERENCE\_NODE
- ENTITY\_NODE
- PROCESSING\_INSTRUCTION\_NODE
- COMMENT\_NODE
- DOCUMENT\_NODE
- DOCUMENT\_TYPE\_NODE
- DOCUMENT\_FRAGMENT\_NODE
- NOTATION\_NODE

## DOMException Types

The following lists the DOMException types:

- INDEX\_SIZE\_ERR
- DOMSTRING\_SIZE\_ERR
- HIERARCHY\_REQUEST\_ERR
- WRONG\_DOCUMENT\_ERR
- INVALID\_CHARACTER\_ERR
- NO\_DATA\_ALLOWED\_ERR
- NO\_MODIFICATION\_ALLOWED\_ERR
- NOT\_FOUND\_ERR
- NOT\_SUPPORTED\_ERR
- INUSE\_ATTRIBUTE\_ERR

## DOM Interface Types

The following lists the DOM Interface types:

- DOM interface types
- DOMNode

- DOMNamedNodeMap
- DOMNodeList
- DOMAttr
- DOMCDATASection
- DOMCharacterData
- DOMComment
- DOMDocumentFragment
- DOMElement
- DOMEntity
- DOMEntityReference
- DOMNotation
- DOMProcessingInstruction
- DOMText
- DOMImplementation
- DOMDocumentType
- DOMDocument

---

## XML SQL Utility (XSU) Specifications and Cheat Sheets

This appendix contains the following sections:

- [Installing XML SQL Utility](#)
- [Requirements for Running XML SQL Utility](#)
- [XML SQL Utility \(XSU\) for Java, Cheat Sheets](#)
- [XML SQL Utility \(XSU\) for PL/SQL, Cheat Sheets](#)

## Installing XML SQL Utility

### Contents of the XSU Distribution

[Table H-1](#) lists XML SQL Utility (XSU) distribution archive (zip file) contents.

**Table H-1 XSU Distribution Contents**

File (with relative location)	Description
relnotes.html	The release notes
env.csh	This files is a helper csh shell script which can set up all the environmental variables needed to run the utility correctly. The user must setup the directory information correctly (for example, point to the installed area for the JDK etc.)
env.bat	This file is the same as the env.csh except that it is written for the Windows platform.
lib/oraclexmlsql.jar	The jar file containing all the Java functions for the utility.
lib/xmlparserv2.jar	The Oracle XML parser V2 packaged with the utility.
lib/oraclexmlsqlload.csh (Unix) lib/oraclexmlsqlload.bat (Windows)	A csh and bat script to help load the utility into an Oracle database. These scripts call loadjava to load the jar file into the database and then run the xmlgenpkg.sql to create the PL/SQL front-end wrappers.
lib/xmlgenpkg.sql	This file contains the sql script for creating the PL/SQL front-end wrappers.

### Installing XML SQL Utility: Procedure

To install XML SQL Utility (XSU) follow these steps:

1. Requirements. Check that you have the correct software requirements loaded.
2. Extract the XSU files
3. Set Up Your Environment Correctly: Client Side
  - CLASSPATH Settings
  - Ensure the database is up
4. Set Up Your Environment Correctly: Server Side

## Installing XSU Downloaded from OTN

Download the correct XSU distribution archive from the Oracle Technology Network web-site (<http://otn.oracle.com>). Expand the downloaded archive. Depending on the usage scenario, perform the following install tasks:

To use the XSU's client side front-end or its java API, you need to:

1. Setup the environment (that is, set CLASSPATH...):
  - **Unix users:** make sure that the path names in `env.csh` are correct; source the `env.csh`. If you are using a shell other than `csh` or `tcsch`, you will have to edit the file to use your shell's syntax.
  - **Windows users:** make sure that the path names in `env.bat` are correct; execute the file.

To use XSU's PL/SQL API, or write java stored procedures on top of XSU's java API, you need to:

1. Confirm that the `USER_PASSWORD` macro in `xsuload.xxx` names the desired schema into which the XSU is to be loaded (default "scott/tiger").
  - **Unix users:** look into `xsuload.csh`
  - **Windows users:** look into `xsuload.bat`
2. Confirm that the Oracle DB into which you are planning to load the XSU is up and java enabled.
3. Execute the appropriate `xsuload.xxx` file. This will:
4. Load Oracle's XML parser for Java into the database. If the parser is already loaded into the database, you can comment out the line in `xsuload.xxx` that loads the parser.
5. Load XSU Java classes (that is, load `xsu12.jar` or `xsu111.jar`). Load the XSU PL/SQL API (that is, execute the `dbmsxsu.sql` PL/SQL script)

## Requirements for Running XML SQL Utility

There are two versions of the utility, `xsu111.jar` and `xsu12.jar`, one compatible for JDK 1.1.x and the other with JDK1.2 respectively.

XML SQL Utility (XSU) is packaged with Oracle8i (8.1.7 and later) and Oracle. XSU is made up of three files:

- `ORACLE_HOME/rdbms/jlib/xsu12.jar` -- Contains all the Java classes which make up XSU. `xsu12` requires JDK1.2.x and JDBC2.x. This is the XSU version loaded into Oracle.
- `ORACLE_HOME/rdbms/jlib/xsu111.jar` -- Contains the same classes as `xsu12.jar`, except that `xsu111` requires JDK1.1.x and JDBC1.x.
- `ORACLE_HOME/rdbms/admin/dbmsxsu.sql` -- This is the SQL script that builds the XSU PL/SQL API. `xsu12.jar` needs to be loaded into the database before `dbmsxsu.sql` is executed.

By default the Oracle installer installs XSU on your hard drive in the locations specified above. It also loads it into the database.

If during initial installation you choose to not install XSU, you can install it later, but the installation becomes less simple. To install XSU later, first install XSU and its dependent components on your system. You can accomplish this using Oracle Installer. Next perform the following steps:

1. If you have not yet loaded XML Parser for Java in the database, go to `ORACLE_HOME/xdk/lib`. Here you will find `xmlparserv2.jar` that you need to load into the database. To do this, see “Loading JAVA Classes” in the *Oracle9i Java Stored Procedures Developer’s Guide*
2. Go to `ORACLE_HOME/admin` and execute `catxsu.sql`

---

---

**Note:** XML SQL Utility (XSU) is also available on OTN at:  
<http://otn.oracle.com/tech/xml> Check here for XSU updates.

---

---

## XSU Requirements

Before installing the utility make sure that you choose the right version of the utility depending on your particular needs. For example, if you can only use the JDK1.1.x version, then download the `xsu111.jar` file. Ensure that you have the JDK and the JDBC drivers correctly downloaded and installed, if not already available.

## Extract the XSU Files

After downloading the zip file, simply extract the contents to a directory of your choice, say `C:\xml`. The files will get expanded in to a subdirectory called `xsu111` or `xsu112` depending on the version of the utility.

## XML SQL Utility (XSU) for Java, Cheat Sheets

The following tables summarize XSU Java API classes and members:

- [Table H-2, "XSU Java API: Class OracleXMLQuery"](#)
- [Table H-3, "XSU Java API: Class OracleXMLSave"](#)
- [Table H-4, "XSU Java API: Class OracleXMLSQLException"](#)
- [Table H-5, "XSU Java API: Class OracleXMLSQLNoRowsException"](#)

**Table H-2 XSU Java API: Class OracleXMLQuery**

Methods, Parameters, Returns, Constructors,....	Description
<b>Class</b> OracleXMLQuery public class OracleXMLQuery extends java.lang.Object where java.lang.Object is oracle.xml.sql.query.OracleXMLQuery	Generates XML from the database given an SQL query.
<b>Fields</b>	-
DTD public static final int DTD	Specifies that the DTD is to be generated.
ERROR_TAG public static final java.lang.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.
MAXROWS_DEFAULT public static final int MAXROWS_DEFAULT	Deprecated since v2.0. Use MAXROWS_ALL instead.
MAXROWS_NONE public static final int MAXROWS_NONE	Deprecated since v2.0. Use 0 instead.
NONE public static final int NONE	Specifies that no DTD is to be generated.
ROW_TAG public static final java.lang.String ROW_TAG	Specifies the default tag name for the ROW elements.
ROWIDATTR_TAG public static final java.lang.String ROWIDATTR_TAG	Specifies the default tag name for the ROW elements.
ROWSET_TAG public static final java.lang.String ROWSET_TAG	Specifies the default tag name for the document.
SCHEMA public static final int SCHEMA	Specifies that no XML schema is to be generated.

**Table H-2 XSU Java API: Class OracleXMLQuery (Cont.)**

<b>Methods, Parameters, Returns, Constructors,....</b>	<b>Description</b>
SKIPROWS_ALL public static final int SKIPROWS_ALL	Specifies that all rows be skipped in the result.
SKIPROWS_DEFAULT public static final int SKIPROWS_DEFAULT	Deprecated since XSU v2.0. Use 0 instead.
SKIPROWS_NONE public static final int SKIPROWS_NONE	Deprecated since XSU v2.0. Use 0 instead.
<b>Constructors</b>	-
OracleXMLQuery(Connection, ResultSet) public OracleXMLQuery(java.sql.Connection conn, java.sql.ResultSet rset)	Constructor for the OracleXMLQueryObject.
Parameters: conn - database connection, rset - jdbc result set object	-
OracleXMLQuery(Connection, String) public OracleXMLQuery(java.sql.Connection conn, java.lang.String query)	Constructor for the OracleXMLQueryObject.
Parameters: conn - database connection, query - the SQL query string	-
OracleXMLQuery(OracleXMLDataSet) public OracleXMLQuery(oracle.xml.sql.dataset.OracleXMLDataSet dset)	Constructor for the OracleXMLQueryObject.
Parameters: conn - database connection, dset - dataset	-
<b>Methods</b>	-
close() public void close()	Closes any open resource, created by the OracleXML engine. This will not close for instance resultset supplied by the user.
getNumRowsProcessed() public long getNumRowsProcessed() Returns: Number of rows processed.	Returns the number of rows processed. -
getXML(OracleXMLDocGen, boolean) public void getXML(oracle.xml.sql.docgen.OracleXMLDocGen doc, boolean withDTD)	Deprecated since XSU v2.0.
getXMLDOM() public org.w3c.dom.Document getXMLDOM() Returns: The DOM representation of the XML document	Transforms the object-relational data, specified in the constructor, into a XML document. -



**Table H-2 XSU Java API: Class OracleXMLQuery (Cont.)**

Methods, Parameters, Returns, Constructors,....	Description
getXMLDOM(boolean) public org.w3c.dom.Document getXMLDOM(boolean withDTD)	Deprecated since XSU 1.2.1. Use getXMLDOM(int) instead.
getXMLDOM(int) public org.w3c.dom.Document getXMLDOM(int metaType)	Transforms the object-relational data, specified in the constructor, into a XML document. 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.
Parameters: metaType - the type of XML metadata (NONE, SCHEMA)	-
Returns: The string representation of the XML document	-
getXMLDOM(Node) public org.w3c.dom.Document getXMLDOM(org.w3c.dom.Node root)	Transforms the object-relational data, specified in the constructor, into XML. If not NULL, the root argument, is considered the "root" element of the XML doc.
Parameters: root - root node to which to append the new XML, Returns: String representation of the XML document	-
getXMLDOM(Node, int) public org.w3c.dom.Document getXMLDOM(org.w3c.dom.Node root, int metaType)	Transforms the object-relational data, specified in the constructor, into XML. If not NULL, the root argument, is considered the "root" element of the XML doc. 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.
Parameters: root - root node to which to append the new XML, metaType - the type of XML metadata (NONE, SCHEMA)	-
Returns: The string representation of the XML document	-
getXMLMetaData(int, boolean) public java.lang.String getXMLMetaData(int metaType, boolean withVer)	Returns the DTD or XMLSchema for the XML document which would have been generated by a getXML call. The "metaType" parameter specifies the type of XML metadata to be generated. The withVer parameter specifies if version header is to be generated or not.

**Table H-2 XSU Java API: Class OracleXMLQuery (Cont.)**

<b>Methods, Parameters, Returns, Constructors,....</b>	<b>Description</b>
Parameters: metaType - XML meta data type to generate (NONE or DTD), withVer - generate the version PI ?	-
getXMLSAX(ContentHandler) public void getXMLSAX(org.xml.sax.ContentHandler sax)	Transforms the object-relational data, specified in the constructor, into an XML document.
Parameters: sax - ContentHandler object to be registered	-
getXMLSchema() public org.w3c.dom.Document getXMLSchema()	Generates the XML Schema(s) corresponding to the specified query.
Returns: the XML Schema(s)	-
getXMLString() public java.lang.String getXMLString()	Transforms the object-relational data, specified in the constructor, into a XML document.
Returns: The string representation of the XML document	-
getXMLString(boolean) public java.lang.String getXMLString(boolean withDTD)	Deprecated since XSU v1.2.1. Use getXMLString(int) instead.
getXMLString(int) public java.lang.String getXMLString(int metaType)	Transforms the object-relational data, specified in the constructor, into a XML document. The metaType argument is used to specify the type of XML metadata the XSU is to generate along with the XML. Valid values for the metaType argument are NONE and DTD (static fields of this class).
Parameters: metaType - Tpe of XML metadata (NONE, DTD, or SCHEMA)	-
Returns: String representation of the XML document	-
getXMLString(Node) public java.lang.String getXMLString(org.w3c.dom.Node root)	Transforms the object-relational data, specified in the constructor, into XML. If not NULL, the root argument, is considered the "root" element of the XML document.
Parameters: root - root node to which to append the new XML	-
Returns: String representation of the XML document	-

**Table H-2 XSU Java API: Class OracleXMLQuery (Cont.)**

Methods, Parameters, Returns, Constructors,....	Description
getXMLString(Node, int) public java.lang.String getXMLString(org.w3c.dom.Node root, int metaType)	Transforms the object-relational data, specified in the constructor, into XML. If not NULL, the root argument, is considered the "root" element of the XML document. MetaType argument specifies the type of XML metadata the XSU is to generate along with the XML. Valid values for the metaType argument are NONE and DTD (static fields of this class). If the root argument is non-null, no DTD is generated even if requested.
Parameters: root - root node to which to append the new XML,	metaType - the type of XML metadata (NONE, DTD, or SCHEMA)
Returns: String representation of the XML document	-
keepCursorState(boolean) public void keepCursorState(boolean alive)	Deprecated since v1.2.1. Use keepObjectOpen instead.
keepObjectOpen(boolean) public void keepObjectOpen(boolean alive)	Default behavior for all the getXML functions which DO NOT TAKE in a ResultSet object is to close the ResultSet object and Statement objects at the end of the call. To use the persistent feature, where by calling getXML repeatedly you get the next set of rows, you need to turn off this behavior by calling this function with value true. That is, OracleXMLQuery would not close the ResultSet and Statement objects after the getXML calls. Call close() to explicitly close the cursor state.
Parameters: alive - keep object open ?	-
removeXSLTParam(String) public void removeXSLTParam(java.lang.String name)	Removes the value of a top-level stylesheet parameter. If no stylesheet is registered, this method does not operate.
Parameters: name - parameter name	-
setCollIdAttr(String) public void setCollIdAttr(java.lang.String collIdAttr)	Deprecated since v1.2.1. Please use setCollIdAttrName instead.

**Table H-2 XSU Java API: Class OracleXMLQuery (Cont.)**

<b>Methods, Parameters, Returns, Constructors,....</b>	<b>Description</b>
setCollIdAttrName(String) public void setCollIdAttrName(java.lang.String attrName)	Sets the name of the id attribute of the collection element's separator tag. Passing null or an empty string for the tag results the row id attribute to be omitted.
Parameters: attrName - attribute name	-
setDataHeader(Reader, String) public void setDataHeader(java.io.Reader header, java.lang.String docTag)	Sets the xml data header. The data header is an XML entity which is appended at the beginning of the query-generated xml entity (ie. rowset). The two entities are enclosed by the tag specified via the docTag argument. The last data header specified is the one that is used. Also, passing in null for the header, parameter unsets the data header.
Parameters: header - header, tag - tag used to enclose the data header and the rowset	-
setDateFormat(String) public void setDateFormat(java.lang.String mask)	Sets the format of the generated dates in the XML doc. The syntax of the date format pattern (i.e. 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.
Parameters: mask - the date mask	-
setEncoding(String) public void setEncoding(java.lang.String enc)	Sets the encoding in the XML doc. If null or an empty string are specified as the encoding, then the default character set is specified in the encoding PI.
Parameters: enc - character set encoding of the XML document	-
setErrorTag(String) public void setErrorTag(java.lang.String tag)	Sets the tag to be used to enclose the XML error documents.
Parameters: tag - tag name	-
setException(Exception) public void setException(java.lang.Exception e)	Allows the user to pass in an exception, and have the XSU handle it.
Parameters: e - the exception to be processed by the XSU.	-

**Table H-2 XSU Java API: Class OracleXMLQuery (Cont.)**

Methods, Parameters, Returns, Constructors,....	Description
setMaxRows(int) public void setMaxRows(int rows)	Sets the max number of rows to be converted to XML. By default there is no max set. To explicitly specify no max see MAXROWS_ALL.
Parameters: rows - max number of rows to generate	-
setMetaHeader(Reader) public void setMetaHeader(java.io.Reader header)	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.
Parameters: header - header	-
setRaiseException(boolean) public void setRaiseException(boolean flag)	Tells the XSU to throw the raised exceptions. If this call is not made or if false is passed to the flag argument, the XSU catches the SQL exceptions and generates an XML document out of the exception's message.
Parameters: flag - throw raised exceptions?	-
setRaiseNoRowsException(boolean) public void setRaiseNoRowsException(boolean flag)	Tells XSU to throw or not to throw an OracleXMLNoRowsException in the case when for one reason or another, the XML document generated is empty. By default, the exception is not thrown.
Parameters: flag - throw OracleXMLNoRowsException if no data found?	-
setRowIdAttrName(String) public void setRowIdAttrName(java.lang.String attrName)	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.
Parameters: attrName - attribute name	-

**Table H-2 XSU Java API: Class OracleXMLQuery (Cont.)**

<b>Methods, Parameters, Returns, Constructors,....</b>	<b>Description</b>
setRowIdAttrValue(String) public void setRowIdAttrValue(java.lang.String colName)	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 (i.e. 0, 1, 2 and so on).
Parameters: colName - column whose value is to be assigned to the row id attr	-
setRowIdColumn(String) public void setRowIdColumn(java.lang.String colName)	Deprecated since XSU v1.2.1. Use setRowIdAttrValue instead.
setRowsetTag(String) public void setRowsetTag(java.lang.String tag)	Sets the tag to be used to enclose the xml dataset.
Parameters: tag - tag name	-
setRowTag(String) public void setRowTag(java.lang.String tag)	Sets the tag to be used to enclose the xml element corresponding to a database record.
Parameters: tag - tag name	-
setSkipRows(int) public void setSkipRows(int rows)	Sets the number of rows to skip. By default 0 rows are skipped. To skip all the rows use SKIPROWS_ALL.
Parameters: rows - number of rows to skip	-
setStyleSheet(String) public void setStyleSheet(java.lang.String uri)	Deprecatet since XSU2.0. Use setStylesheetHeader instead.
setStyleSheet(String, String) public void setStyleSheet(java.lang.String uri, java.lang.String type)	Deprecated since XSU2.0. Use setStylesheetHeader instead.
setStylesheetHeader(String) public void setStylesheetHeader(java.lang.String uri)	Sets the stylesheet header (that is, stylesheet processing instructions) in the generated XML doc. Passing null for the uri argument will unset the stylesheet header and the stylesheet type.
Parameters: uri - stylesheet URI	-

**Table H-2 XSU Java API: Class OracleXMLQuery (Cont.)**

Methods, Parameters, Returns, Constructors,....	Description
setStylesheetHeader(String, String) public void setStylesheetHeader(java.lang.String uri, java.lang.String type)	Sets the stylesheet header (that is, stylesheet processing instructions) in the generated XML document. Passing null for the URI argument will unset the stylesheet header and the stylesheet type.
Parameters: uri - stylesheet URI, type - stylesheet type; defaults to 'text/xsl'	-
setXSLT(Reader, String) public void setXSLT(java.io.Reader stylesheet, java.lang.String ref)	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.
Parameters: stylesheet - the stylesheet, ref - URL for include, import and external entities	-
setXSLT(String, String) public void setXSLT(java.lang.String stylesheet, java.lang.String ref)	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.
Parameters: stylesheet - the stylesheet URI, ref - URL for include, import and external entities	-
setXSLTParam(String, String) public void setXSLTParam(java.lang.String name, java.lang.String value)	Sets the value of a top-level stylesheet parameter. The parameter value is expected to be a valid XPath expression (String literal values would therefore have to be explicitly quoted). If no stylesheet is registered, this method is not operational.
Parameters: name - parameter name, value - parameter value as an XPATH expression	-
useLowerCaseTagNames() public void useLowerCaseTagNames()	Sets the case to be lower for all tag names. Make this call after all the desired tags have been set.

**Table H–2 XSU Java API: Class OracleXMLQuery (Cont.)**

Methods, Parameters, Returns, Constructors,....	Description
useNullAttributeIndicator(boolean) public void useNullAttributeIndicator(boolean flag)	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.
Parameters: flag - use attribute to indicate null?	-
useTypeForCollElemTag(boolean) public void useTypeForCollElemTag(boolean flag)	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 XSU to use the collection element's type name as the collection element tag name.
Parameters: flag - use the coll. elem. type as its tag name?	-
useUpperCaseTagNames() public void useUpperCaseTagNames()	Sets the case to be upper for all tag names. Make this call after all the desired tags have been set.

**Table H–3 XSU Java API: Class OracleXMLSave**

Methods, Parameters, Returns, Constructors, ...	Description
<b>Class</b> OracleXMLSave public class OracleXMLSave extends java.lang.Object where java.lang.Object is oracle.xml.sql.dml.OracleXMLSave	Supports canonical mapping from XML to object-relational tables or views. It supports inserts, updates and deletes. You first create the class by passing in the table name on which the DML operations need to be done. After that, the user is free to use the insert/update/delete on this table. The useful functions provided in this class help identify the key columns for update or delete and restrict the columns being updated.
<b>Fields</b>	-
DATE_FORMAT public static final java.lang.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



**Table H-3 XSU Java API: Class OracleXMLSave (Cont.)**

Methods, Parameters, Returns, Constructors, ...	Description
<b>Constructors</b>	-
OracleXMLSave(Connection, String) public OracleXMLSave(java.sql.Connection oconn, java.lang.String tableName)	The public constructor for the Save class.
Parameters	oconn - Connection object (connection to the database), tableName - The name of the table that should be updated
<b>Methods</b>	-
cleanLobList() public void cleanLobList()	-
close() public void close()	Closes/deallocates all the context associated with this object.
createUrl(String) public java.net.URL createURL(java.lang.String fileName)	Deprecated since XSU2.0. Use the static version of this method instead.
deleteXML(Document) public int deleteXML(org.w3c.dom.Document doc)	Deletes the rows in the table based on the XML document.
Parameters	xmlDoc - The XML document in DOM form
Returns	The number of XML ROW elements processed. See Also: deleteXML(URL)
deleteXML(InputStream) public int deleteXML(java.io.InputStream xmlStream)	Deletes the rows in the table based on the XML document.
Parameters	xmlDoc - The XML document in Stream form
Returns	The number of XML ROW elements processed. See Also: deleteXML(URL)
deleteXML(Reader) public int deleteXML(java.io.Reader xmlStream)	Deletes the rows in the table based on the XML document.
Parameters	xmlDoc - The XML document in Stream form
Returns	The number of XML ROW elements processed. See Also: deleteXML(URL)

**Table H-3 XSU Java API: Class OracleXMLSave (Cont.)**

<b>Methods, Parameters, Returns, Constructors, ...</b>	<b>Description</b>
<code>deleteXML(String)</code> <code>public int deleteXML(java.lang.String xmlDoc)</code>	Deletes the rows in the table based on the XML document.
Parameters	xmlDoc - The XML document in String form
Returns	The number of XML ROW elements processed. See Also: <code>deleteXML(URL)</code>
<code>deleteXML(URL)</code> <code>public int deleteXML(java.net.URL url)</code>	Deletes rows from a specified table based on the element values in the supplied XML document. 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.
Parameters	url - The URL to the document to use to delete the rows in the table
Returns	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.
<code>finalize()</code> <code>protected void finalize()</code>	Overrides: <code>java.lang.Object.finalize()</code> in class <code>java.lang.Object</code>
<code>getURL(String)</code> <code>public static java.net.URL getURL(java.lang.String target)</code>	Given a file name or a URL it return a URL object. If the argument passed is not in the valid URL format (e.g. <code>http://..</code> or <code>file://</code> ) then this method tried to fix the argument by pre-pending "file://" to the argument. If a null or an empty string are passed to it, null is returned.
Parameters	target - file name or URL string
Returns	the URL object identifying the target entity

**Table H-3 XSU Java API: Class OracleXMLSave (Cont.)**

Methods, Parameters, Returns, Constructors, ...	Description
insertXML(Document) public int insertXML(org.w3c.dom.Document doc)	-
insertXML(InputStream) public int insertXML(java.io.InputStream xmlStream)	-
insertXML(Reader) public int insertXML(java.io.Reader xmlStream)	-
insertXML(String) public int insertXML(java.lang.String xmlDoc)	-
insertXML(URL) public int insertXML(java.net.URL url)	Inserts an XML document from a specified URL into the specified table. By default, the insert routine inserts values into the table by matching the element name with the column name and inserts a null value for all elements missing in the input document. By setting the list of columns to insert using the <code>setUpdateColumnList()</code> you can restrict the insert to only insert values into those columns and let the default values for other columns to be inserted. For more details see <a href="#">Chapter 7, "XML SQL Utility (XSU)"</a> and <i>Oracle9i XML Reference</i>
Parameters	url - The URL to the document to use to insert rows into the table
Returns	The number of rows inserted.
removeXSLTParam(String) public void removeXSLTParam(java.lang.String name)	Removes the value of a top-level stylesheet parameter. If no stylesheet is registered, this method is non operational.
Parameters	name - parameter name

**Table H-3 XSU Java API: Class OracleXMLSave (Cont.)**

Methods, Parameters, Returns, Constructors, ...	Description
setBatchSize(int) public void setBatchSize(int size)	Changes the batch size used during DML operations. When inserting, updating, or deleting, it is better to batch the operations so that the database can execute it once rather than as separate statements. However, more memory is needed to hold all the bind values before the operation is done. Note when batching is used, the commits occur only in terms of batches. So if one of the statement inside a batch fails, the whole batch is rolled back. If this behaviour is unacceptable, set the batch size to 1. The default batch size is <code>DEFAULT_BATCH_SIZE</code> ;
Parameters	size - The batch size to use for all DML
setCommitBatch(int) public void setCommitBatch(int size)	Sets the commit batch size. The commit batch size refers to the number or records inserted after which a commit should follow. If <code>commitBatch</code> is <code>&lt; 1</code> or the session is in "auto-commit" mode then XSU does not make any explicit commit's. By default the commit-batch size is 0.
Parameters	size - commit batch size
setDateFormat(String) public void setDateFormat(java.lang.String mask)	Describes to XSU the format of the dates in the XML document. By default, <code>OracleXMLSave</code> 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 patern (that is, the date mask), should conform to the requirements of the <code>java.text.SimpleDateFormat</code> class. Setting the mask to null or an empty string, results the use of the default mask -- <code>OracleXMLSave.DATE_FORMAT</code> .

**Table H-3 XSU Java API: Class OracleXMLSave (Cont.)**

Methods, Parameters, Returns, Constructors, ...	Description
Parameters	mask - the date mask
setIgnoreCase(boolean) public void setIgnoreCase(boolean ignore)	XSU maps XML elements to database columns/ attributes based on element names (XML tags). This function tells XSU to do this match case insensitively. This resetting of case may affect metadata caching done when creating the Save object.
Parameters	flag - ignore tag case in the XML doc? 0-false 1-true
setKeyColumnList(String[]) public void setKeyColumnList(java.lang.String[] keyColNames)	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 inserts. 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.
Parameters	keyColNames - The names of the list of columns that are used as keys
setRowTag(String) public void setRowTag(java.lang.String rowTag)	Names the tag used in the XML doc., 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.
Parameters	tag - tag name

**Table H-3 XSU Java API: Class OracleXMLSave (Cont.)**

Methods, Parameters, Returns, Constructors, ...	Description
setUpdateColumnList(String[]) public void setUpdateColumnList(java.lang.String[] updColNames)	Sets column values to be updated. Only valid for inserts and updates. Ignored for deletes. For inserts, the default is to insert values to all the columns in the table. For 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 are updated in the UPDATE or INSERT statement. All other elements in the document are ignored.
Parameters	updColNames - The string list of columns to be updated
setXSLT(Reader, String) public void setXSLT(java.io.Reader stylesheet, java.lang.String ref)	Registers an 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.
Parameters	stylesheet - the stylesheet, ref - URL for include, import and external entities
setXSLT(String, String) public void setXSLT(java.lang.String stylesheet, java.lang.String ref)	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.
Parameters	stylesheet - the stylesheet URI, ref - URL for include, import and external entities
setXSLTParam(String, String) public void setXSLTParam(java.lang.String name, java.lang.String value)	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.

**Table H-3 XSU Java API: Class OracleXMLSave (Cont.)**

<b>Methods, Parameters, Returns, Constructors, ...</b>	<b>Description</b>
Parameters	name - parameter name, value - parameter value as an XPATH expression
updateXML(Document) public int updateXML(org.w3c.dom.Document doc)	Updates the table given the XML document in a DOM tree form.
Parameters	xmlDoc - The DOM tree form of the XML document
Returns	The number of XML elements processed. See Also: updateXML(URL)
updateXML(InputStream) public int updateXML(java.io.InputStream xmlStream)	Updates the table given the XML document in a stream form.
Parameters	xmlDoc - The stream form of the XML document
Returns	The number of XML elements processed. See Also: updateXML(URL)
updateXML(Reader) public int updateXML(java.io.Reader xmlStream)	Updates the table given the XML document in a stream form.
Parameters	xmlDoc - The stream form of the XML document
Returns	The number of XML elements processed. See Also: updateXML(URL)
updateXML(String) public int updateXML(java.lang.String xmlDoc)	Updates the table given the XML document in a string form.
Parameters	xmlDoc - The string form of the XML document
Returns	The number of XML elements processed. See Also: updateXML(URL)

**Table H-3 XSU Java API: Class OracleXMLSave (Cont.)**

Methods, Parameters, Returns, Constructors, ...	Description
updateXML(URL) public int updateXML(java.net.URL url)	Updates the columns in a database table, based on the element values in the supplied XML document. 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 and then updates all the columns in the table for which there is an equivalent element present in the XML document.
Parameters	url - The URL to the document to use to update the table
Returns	The number of XML row elements processed. This may or may not be equal to the number of database rows modified based on whether the rows selected through the XML document uniquely identified the rows in the table.

**Table H-4 XSU Java API: Class OracleXMLSQLException**

Constructors and Methods	Description
<b>Class</b>	-
OracleXMLSQLException public class OracleXMLSQLException extends java.lang.RuntimeException	
<b>Constructors</b>	-
OracleXMLSQLException(Exception) public OracleXMLSQLException(java.lang.Exception e)	-
OracleXMLSQLException(Exception, String) public OracleXMLSQLException(java.lang.Exception e, java.lang.String errorTagName)	-



**Table H-4 XSU Java API: Class OracleXMLSQLException (Cont.)**

<b>Constructors and Methods</b>	<b>Description</b>
OracleXMLSQLException(String) public OracleXMLSQLException(java.lang.String message)	-
OracleXMLSQLException(String, Exception) public OracleXMLSQLException(java.lang.String message, java.lang.Exception e)	-
OracleXMLSQLException(String, Exception, String) public OracleXMLSQLException(java.lang.String message, java.lang.Exception e, java.lang.String errorTagName)	-
OracleXMLSQLException(String, int) public OracleXMLSQLException(java.lang.String message, int errorCode)	-
OracleXMLSQLException(String, int, String) public OracleXMLSQLException(java.lang.String message, int errorCode, java.lang.String errorTagName)	-
OracleXMLSQLException(String, String) public OracleXMLSQLException(java.lang.String message, java.lang.String errorTagName)	-
<b>Methods</b>	-
getErrorCode() public int getErrorCode()	-
getParentException() public java.lang.Exception getParentException()	Returns the original exception, if there was one; otherwise, it returns null.
getXMLErrorString() public java.lang.String getXMLErrorString()	Prints the XML error string with the given error tag name.
getXMLSQLExceptionString() public java.lang.String getXMLSQLExceptionString()	Prints the SQL parameters as well in the error message.
setErrorTag(String) public void setErrorTag(java.lang.String tagName)	Sets the error tag name which is then used by getXMLErrorString and getXMLSQLExceptionString, to generate xml error reports.

**Table H-5 XSU Java API: Class OracleXMLSQLNoRowsException**

Constructors	Description
<b>Class</b>	-
OracleXMLSQLNoRowsException public class OracleXMLSQLNoRowsException extends OracleXMLSQLException	
<b>Constructors</b>	-
OracleXMLSQLNoRowsException() public OracleXMLSQLNoRowsException()	-
OracleXMLSQLNoRowsException(String) public OracleXMLSQLNoRowsException(java.lang.String errorTag)	-

## XML SQL Utility (XSU) for PL/SQL, Cheat Sheets

XML SQL Utility (XSU) for PL/SQL offers the following PL/SQL packages:

- DBMS\_XMLQuery -- provides database-to-XML functionality.
- DBMS\_XMLSave -- provides XML-to-database functionality.

### DBMS\_XMLQuery PL/SQL Package

[Table H-6](#) lists DBMS\_XMLQuery procedures, functions, and constants.

**Table H-6 DBMS\_XMLQuery Procedures, Functions, Types, and Constants**

<b>PROCEDURE (Unless Noted Otherwise)</b>	<b>Description</b>
TYPE: ctxType	Type of query context handle. This is the return type of "DBMS_XMLQuery.newContext()".
<b>CONSTANTS</b>	
DEFAULT_ROWSETTAG	Mostly this is the root node tag name,ROWSET
DEFAULT_ERRORTAG	ERROR
DEFAULT_ROWIDATTR	NUM
DEFAULT_ROWTAG	ROW
DEFAULT_DATE_FORMAT	'MM/dd/yyyy HH:mm:ss'
ALL_ROWS	All rows are needed in the output
NONE	For example, no DTD
DTD	DTD generation required
LOWER_CASE	Use lower case tags
UPPER_CASE	User upper case tags
closeContext(ctxType)	Closes/deallocates a particular query context
FUNCTION: getDTD(ctxType, BOOLEAN := false)	-
getDTD(ctxType, CLOB, BOOLEAN := false)	Generates the DTD based on the SQL query used to init.
getExceptionContent(ctxType, NUMBER, VARCHAR2)	-
FUNCTION: getXML(ctxType, NUMBER := NONE)	-
getXML(ctxType, CLOB, NUMBER := NONE)	Generates the XML document.
FUNCTION: newContext(VARCHAR2) --> RETURN -- ctxType	Creates a query context, and it returns the context handle.
FUNCTION: newContext(CLOB) ---> RETURN ---> ctxType	Creates a query context, and it returns the context handle.
propagateOriginalException(ctxType, BOOLEAN)	Tells XSU if an exception is raised, and is being thrown, that XSU should throw the exception raised; rather then, wrapping it with an OracleXMLSQLException.

**Table H-6 DBMS\_XMLQuery Procedures, Functions, Types, and Constants (Cont.)**

<b>PROCEDURE (Unless Noted Otherwise)</b>	<b>Description</b>
setBindValue(ctxType, VARCHAR2, VARCHAR2)	Sets a value for a particular bind name.
setCollIdAttrName(ctxType, VARCHAR2)	Sets the name of the id attribute of the collection element's separator tag.
setDataHeader(ctxType, CLOB := null, VARCHAR2 := null)	Sets the XML data header.
setDateFormat(ctxType, VARCHAR2)	Sets the format of the generated dates in the XML document.
setErrorTag(ctxType, VARCHAR2)	Sets the tag to be used to enclose the xml error docs.
setMaxRows (ctxType, NUMBER)	Sets the max number of rows to be converted to XML.
setMetaHeader(ctxType, CLOB := null)	Sets the XML meta header.
setRaiseException(ctxType, BOOLEAN)	Tells the XSU to throw the raised exceptions.
setRaiseNoRowsException(ctxType, BOOLEAN)	Tells the XSU to throw or not to throw an OracleXMLNoRowsException in the case when for one reason or another, the XML document generated is empty.
setRowIdAttrName(ctxType, VARCHAR2)	Sets the name of the id attribute of the row enclosing tag.
setRowIdAttrValue(ctxType, VARCHAR2)	Specifies the scalar column whose value is to be assigned to the id attribute of the row enclosing tag.
setRowsetTag(ctxType, VARCHAR2)	Sets the tag to be used to enclose the XML dataset.
setRowTag(ctxType, VARCHAR2)	Sets the tag to be used to enclose the xml element corresponding to a database.
setSkipRows(ctxType, NUMBER)	Sets the number of rows to skip.
setStylesheetHeader(ctxType, VARCHAR2, VARCHAR2 := 'text/xml')	Sets the stylesheet header
setTagCase(ctxType, NUMBER)	Specified the case of the generated XML tags.

**Table H-6 DBMS\_XMLQuery Procedures, Functions, Types, and Constants (Cont.)**

PROCEDURE (Unless Noted Otherwise)	Description
setXSLT(ctxType, VARCHAR2, VARCHAR2 := null)	Registers a stylesheet to be applied to generated XML.
setXSLT(ctxType, CLOB, VARCHAR2 := null)	Registers a stylesheet to be applied to generated XML.
useNullAttributeIndicator(ctxType, BOOLEAN)	Specified whether to use an XML attribute to indicate NULLness, or to do it by omitting the inclusion of the particular entity in the XML document.

## DBMS\_XMLSave PL/SQL Package

[Table H-7](#) lists DBMS\_XMLSave procedures, functions, types, and constants.

**Table H-7 DBMS\_XMLSave Procedures, Functions, Types, and Constants**

PROCEDURE (Unless Noted Otherwise)	Description
TYPE: ctxType	Type of query context handle. The return type of "DBMS_XMLSave.newContext()".
CONSTANTS:	
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
clearKeyColumnList(ctxType)	Clears the key column list.
clearUpdateColumnList(ctxType)	Clears the update column list.
closeContext(ctxType)	Closes/deallocates a particular save context
FUNCTION: deleteXML(ctxType, CLOB)	Deletes records specified by data from the XML document, from the table specified at the context creation time.
RETURN	
NUMBER	

**Table H-7 DBMS\_XMLSave Procedures, Functions, Types, and Constants(Cont.)**

<b>PROCEDURE (Unless Noted Otherwise)</b>	<b>Description</b>
FUNCTION: deleteXML(ctxType, VARCHAR2) RETURN NUMBER	Deletes records specified by data from the XML document, from the table specified at the context creation time.
getExceptionContent(ctxType, NUMBER, VARCHAR2)	Via its arguments, this method returns the thrown exception's error code and error message
FUNCTION: insertXML(ctxType, CLOB) RETURN NUMBER	Inserts the XML document into the table specified at the context creation time.
FUNCTION: insertXML(ctxType, VARCHAR2) RETURN NUMBER .	Inserts the XML document into the table specified at the context creation time
FUNCTION: newContext(targetTable IN VARCHAR2) RETURN ctxType	Creates a save context, and it returns the context handle.
propagateOriginalException(ctxType, BOOLEAN)	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.
setBatchSize(ctxType, NUMBER)	Changes the batch size used during DML operations.
setCommitBatch(ctxType, NUMBER)	Sets the commit batch size.
setDateFormat(ctxType, VARCHAR2)	Describes to the XSU the format of the dates in the XML document.
setIgnoreCase(ctxType, NUMBER)	XSU maps XML elements to the database.
setKeyColumn(ctxType, VARCHAR2)	Adds a column to the "key column list".
setRowTag(ctxType, VARCHAR2)	Names the tag used in the XML document, to enclose the XML elements corresponding to the database.
setUpdateColumn(ctxType, VARCHAR2)	Adds a column to the "update column list".
getExceptionContent(ctxType, NUMBER, VARCHAR2)	Updates the table specified at the context creation time with data from the XML document.
propagateOriginalException(ctxType, BOOLEAN)	Updates the table specified at the context creation time with data from the XML document.

**Table H-7 DBMS\_XMLSave Procedures, Functions, Types, and Constants(Cont.)**

<b>PROCEDURE (Unless Noted Otherwise)</b>	<b>Description</b>
FUNCTION: newContext(targetTable IN VARCHAR2)	-
RETURN ctxType	Creates a save context, and it returns the context handle.
Parameter	targetTable Target table to load XML document to.
Returns:	The context handle.
closeContext(ctxHdl IN ctxType)	Closes/deallocates a particular save context.
Parameter ctxHdl - Context handle	-
setRowTag(ctxHdl IN ctxType, tag IN VARCHAR2)	Names the tag used in the XML document, to enclose the XML elements corresponding to the database records.
Parameters	ctxHdl - Context handle, tag - Tag name
setIgnoreCase(ctxHdl IN ctxType, flag IN NUMBER)	XSU maps XML elements to the database columns/attributes based on element names (XML tags). This function tells XSU to do this match case insensitive.
Parameters	ctxHdl- context handle, flag - ignore tag case in the XML document? 0-false 1-true
setDateFormat(ctxHdl IN ctxType, mask IN VARCHAR2)	Describes to 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 <code>java.text.SimpleDateFormat</code> class. Setting the mask to null or an empty string, results the use of the default mask -- <code>OracleXMLCore.DATE_FORMAT</code> .
Parameters ctxHdl - Context handle, mask - Date mask	-

**Table H-7 DBMS\_XMLSave Procedures, Functions, Types, and Constants(Cont.)**

<b>PROCEDURE (Unless Noted Otherwise)</b>	<b>Description</b>
setBatchSize(ctxHdl IN ctxType, batchSize IN NUMBER);	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. 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 behaviour is unacceptable, set the batch size to 1. See Also: DEFAULT_BATCH_SIZE
Parameters	ctxHdl - Context handle, batchSize - Batch size
setCommitBatch(ctxHdl IN ctxType, batchSize IN NUMBER);	Sets the commit batch size. Commit batch size refers to the number of records inserted after which a commit should follow. If commitBatch is < 1 or the session is in "auto-commit" mode then XSU does not make any explicit commit's. By default the commit-batch size is 0.
Parameters	ctxHdl - Context handle, ParambatchSize - Commit batch size
setUpdateColumn(ctxHdl IN ctxType, colName IN VARCHAR2);	Adds a column to the "update column list". In inserts, the default is to insert values to all the columns in the table. For 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.
Parameters	ctxHdl - Context handle, colName - Column to be added to the update column list
clearUpdateColumnList(ctxHdl IN ctxType)	Clears the update column list. See Also: setUpdateColumn
Parameters	ctxHdl - Context handle



**Table H-7 DBMS\_XMLSave Procedures, Functions, Types, and Constants(Cont.)**

<b>PROCEDURE (Unless Noted Otherwise)</b>	<b>Description</b>
setKeyColumn(ctxHdl IN ctxType, colName IN VARCHAR2)	Adds a column to the "key column list". In 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.
Parameters	ctxHdl - Context handle, colName - Column to be added to the key column list
clearKeyColumnList(ctxHdl IN ctxType)	Clears the key column list. See Also: setKeyColumn
Parameters	ctxHdl - Context handle
FUNCTION insertXML(ctxHdl IN ctxType, xDoc IN VARCHAR2)	-
RETURN NUMBER	Inserts the XML document into the table specified at the context creation time.
Parameters	ctxHdl - Context handle, xDoc - String containing the XML document
Returns	The number of rows inserted.
FUNCTION insertXML(ctxHdl IN ctxType, xDoc IN CLOB)	-
RETURN NUMBER	Inserts the XML document into the table specified at the context creation time.
Parameters	ctxHdl - Context handle, xDoc - String containing the XML document
Returns	The number of rows inserted.
FUNCTION updateXML(ctxHdl IN ctxType, xDoc IN VARCHAR2)	-
RETURN NUMBER	Updates the table specified at the context creation time with data from the XML document.
Parameters	ctxHdl - Context handle, xDoc - String containing the XML document
Returns	The number of rows updated.
FUNCTION updateXML(ctxHdl IN ctxType, xDoc IN CLOB)	-
RETURN NUMBER	Updates the table specified at the context creation time with data from the XML document.

**Table H-7 DBMS\_XMLSave Procedures, Functions, Types, and Constants(Cont.)**

<b>PROCEDURE (Unless Noted Otherwise)</b>	<b>Description</b>
Parameters	ctxHdl - context handle, xDocl - string containing the XML document
Returns	The number of rows updated.
FUNCTION deleteXML(ctxHdl IN ctxType, xDoc IN VARCHAR2)	-
RETURN NUMBER	Deletes records specified by data from the XML document, from the table specified at the context creation time.
Parameters	ctxHdl - context handle, xDoc - string containing the XML document
Returns	The number of rows deleted.
FUNCTION deleteXML(ctxHdl IN ctxType, xDoc IN CLOB)	-
RETURN NUMBER	Deletes records specified by data from the XML document, from the table specified at the context creation time.
Parameters	ctxHdl - context handle, xDocl - string containing the XML document
Returns	The number of rows deleted.
propagateOriginalException(ctxHdl IN ctxType, flag IN BOOLEAN)	Tells XSU that if an exception is raised, and is being thrown, XSU should throw the exception raised; rather than, wrapping it with OracleXMLSQLException.
Parameters	ctxHdl - Context handle, flag - Propagate original exception? 0-false 1-true
getExceptionContent(ctxHdl IN ctxType, errNo OUT NUMBER, errMsg OUT VARCHAR2)	Via its arguments, this 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; PL/SQL is unable to access the original exception.
Parameters	ctxHdl - Context handle, errNo - Error number, errMsg - Error message

**See Also:**

- *Oracle9i XML Reference*
- [Chapter 7, "XML SQL Utility \(XSU\)"](#)
- <http://otn.oracle.com/tech/xml>



---

# Glossary

## **API**

Application Program Interface. See application program, definition interface.

## **application program interface (API)**

A set of public programmatic interfaces that consist of a language and message format to communicate with an operating system or other programmatic environment, such as databases, Web servers, JVMs, and so forth. These messages typically call functions and methods available for application development.

## **application server**

A server designed to host applications and their environments, permitting server applications to run. A typical example is OAS, which is able to host Java, C, C++, and PL/SQL applications in cases where a remote client controls the interface. See also Oracle Application Server.

## **attribute**

A property of an element that consists of a name and a value separated by an equals sign and contained within the start tags after the element name. In this example, `<Price units='USD'>5</Price>`, `units` is the attribute and `USD` is its value, which must be in single or double quotes. Attributes may reside in the document or DTD. Elements may have many attributes but their retrieval order is not defined.

## **BC4J**

Business Components for Java.

**Business-to-Business (B2B)**

A term describing the communication between businesses in the selling of goods and services to each other. The software infrastructure to enable this is referred to as an exchange.

**Business-to-Consumer (B2C)**

A term describing the communication between businesses and consumers in the selling of goods and services.

**BFILES**

External binary files that exist outside the database tablespaces residing in the operating system. BFILES are referenced from the database semantics, and are also known as External LOBs.

**Binary Large Object (BLOB)**

A Large Object datatype whose content consists of binary data. Additionally, this data is considered raw as its structure is not recognized by the database.

**BLOB**

See Binary Large Object.

**callback**

A programmatic technique in which one process starts another and then continues. The second process then calls the first as a result of an action, value, or other event. This technique is used in most programs that have a user interface to allow continuous interaction.

**cartridge**

A stored program in Java or PL/SQL that adds the necessary functionality for the database to understand and manipulate a new datatype. Cartridges interface through the Extensibility Framework within Oracle 8 or 8i. interMedia Text is just such a cartridge, adding support for reading, writing, and searching text documents stored within the database.

**CDATA**

See character data.

**CDF**

Channel Definition Format. Provides a way to exchange information about channels on the internet.

**CGI**

See Common Gateway Interface.

**CSS**

Cascading Style Sheets.

**character data (CDATA)**

Text in a document that should not be parsed is put within a CDATA section. This allows for the inclusion of characters that would otherwise have special functions, such as &, <, >, etc. CDATA sections can be used in the content of an element or in attributes.

**Common Gateway Interface (CGI)**

The generic acronym for the programming interfaces enabling Web servers to execute other programs and pass their output to HTML pages, graphics, audio, and video sent to browsers.

**child element**

An element that is wholly contained within another, which is referred to as its parent element. For example `<Parent><Child></Child></Parent>` illustrates a child element nested within its parent element.

**Class Generator**

A utility that accepts an input file and creates a set of output classes that have corresponding functionality. In the case of the XML Class Generator, the input file is a DTD and the output is a series of classes that can be used to create XML documents conforming with the DTD.

**CLASSPATH**

The operating system environmental variable that the JVM uses to find the classes it needs to run applications.

**client-server**

The term used to describe the application architecture where the actual application runs on the client but accesses data or other external processes on a server across a network.

**Character Large Object (CLOB)**

The LOB datatype whose value is composed of character data corresponding to the database character set. A CLOB may be indexed and searched by the interMedia Text search engine.

**CLOB**

See Character Large Object.

**command line**

The interface method in which the user enters in commands at the command interpreter's prompt.

**Common Object Request Broker API (CORBA)**

An Object Management Group standard for communicating between distributed objects across a network. These self-contained software modules can be used by applications running on different platforms or operating systems. CORBA objects and their data formats and functions are defined in the Interface Definition Language (IDL), which can be compiled in a variety of languages including Java, C, C++, Smalltalk and COBOL.

**Common Oracle Runtime Environment (CORE)**

The library of functions written in C that provides developers the ability to create code that can be easily ported to virtually any platform and operating system.

**CORBA**

See Common Object Request Broker.

**Database Access Descriptor (DAD)**

A DAD is a named set of configuration values used for database access. A DAD specifies information such as the database name or the SQL\*Net V2 service name, the ORACLE\_HOME directory, and NLS configuration information such as language, sort type, and date language.

**datagram**

A text fragment, which may be in XML format, that is returned to the requester embedded in an HTML page from a SQL query processed by the XSQL Servlet.

**DOCTYPE**

The term used as the tag name designating the DTD or its reference within an XML document. For example, `<!DOCTYPE person SYSTEM "person.dtd">` declares the



root element name as person and an external DTD as person.dtd in the file system. Internal DTDs are declared within the DOCTYPE declaration.

### **Document Object Model (DOM)**

An in-memory tree-based object representation of an XML document that enables programmatic access to its elements and attributes. The DOM object and its interface is a W3C recommendation. It specifies the Document Object Model of an XML Document including the APIs for programmatic access. DOM views the parsed document as a tree of objects.

### **Document Type Definition (DTD)**

A set of rules that define the allowable structure of an XML document. DTDs are text files that derive their format from SGML and can either be included in an XML document by using the DOCTYPE element or by using an external file through a DOCTYPE reference.

### **DOM**

See Document Object Model.

### **DTD**

See Document Type Definition.

### **EDI**

Electronic Data Interchange.

### **Enterprise Java Bean (EJB)**

An independent program module that runs within a JVM on the server. CORBA provides the infrastructure for EJBs, and a container layer provides security, transaction support, and other common functions on any supported server.

### **element**

The basic logical unit of an XML document that may serve as a container for other elements as children, data, attributes, and their values. Elements are identified by start-tags, <name> and end-tags</name> or in the case of empty elements, <name/>.

### **empty element**

An element without text content or child elements. It may only contain attributes and their values. Empty elements are of the form <name/> or <name></name> where there is no space between the tags.

**entity**

A string of characters that may represent either another string of characters or special characters that are not part of the document's character set. Entities and the text that is substituted for them by the parser are declared in the DTD.

**eXtensible Markup Language (XML)**

An open standard for describing data developed by the W3C using a subset of the SGML syntax and designed for Internet use. Version 1.0 is the current standard, having been published as a W3C Recommendation in February 1998.

**eXtensible Stylesheet Language (XSL)**

The language used within stylesheets to transform or render XML documents. There are two W3C recommendations covering XSL stylesheets—XSL Transformations (XSLT) and XSL Formatting Objects (XSLFO).

**XSL**

(W3C) eXtensible Stylesheet Language, XSL consists of two W3C recommendations - XSL Transformations for transforming one XML document into another and XSL Formatting Objects for specifying the presentation of an XML document. XSL is a language for expressing stylesheets. It consists of two parts:

- A language for transforming XML documents (XSLT), and
- An XML vocabulary for specifying formatting semantics (XSL:FO).

An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary.

**eXtensible Stylesheet Language Formatting Object (XSLFO)**

The W3C standard specification that defines an XML vocabulary for specifying formatting semantics.

**eXtensible Stylesheet Language Transformation (XSLT)**

Also written as XSL-T. The XSL W3C standard specification that defines a transformation language to convert one XML document into another.

**HTML**

See Hypertext Markup Language.

**HTTP**

See Hypertext Transport Protocol.

**hypertext**

The method of creating and publishing text documents in which users can navigate between other documents or graphics by selecting words or phrases designated as hyperlinks.

**Hypertext Markup Language (HTML)**

The markup language used to create the files sent to Web browsers and that serves as the basis of the World Wide Web. The next version of HTML will be called xHTML and will be an XML application.

**Hypertext Transport Protocol (HTTP)**

The protocol used for transporting HTML files across the Internet between Web servers and browsers.

**IDE**

See Integrated Development Environment.

**iFS**

See Internet File System.

**Integrated Development Environment (IDE)**

A set of programs designed to aide in the development of software run from a single user interface. JDeveloper is an IDE for Java development as it includes an editor, compiler, debugger, syntax checker, help system, and so on to permit Java software development through a single user interface.

**Internet File System (iFS)**

The Oracle file system and Java-based development environment that either runs inside the Oracle8i database or on a middle tier and provides a means of creating, storing, and managing multiple types of documents in a single database repository.

**Internet Inter-ORB Protocol (IIOP)**

The protocol used by CORBA to exchange messages on a TCP/IP network such as the Internet.

**instantiate**

A term used in object-based languages such as Java and C++ to refer to the creation of an object of a specific class.

**interMedia**

The term used to describe the collection of complex data types and their access within Oracle8i. These include text, video, time-series, and spatial data types.

**Java**

A high-level programming language developed and maintained by Sun Microsystems where applications run in a virtual machine known as a JVM. The JVM is responsible for all interfaces to the operating system. This architecture permits developers to create Java applications and applets that can run on any operating system or platform that has a JVM.

**Java Bean**

An independent program module that runs within a JVM, typically for creating user interfaces on the client. The server equivalent is called an Enterprise Java Bean (EJB). See also Enterprise Java Bean.

**Java Database Connectivity (JDBC)**

The programming API that enables Java applications to access a database through the SQL language. JDBC drivers are written in Java for platform independence but are specific to each database.

**Java Developer's Kit (JDK)**

The collection of Java classes, runtime, compiler, debugger, and usually source code for a version of Java that makes up a Java development environment. JDKs are designated by versions, and Java 2 is used to designate versions from 1.2 onward.

**Java Runtime Environment (JRE)**

The collection of compiled classes that make up the Java virtual machine on a platform. JREs are designated by versions, and Java 2 is used to designate versions from 1.2 onward.

**Java Server Page (JSP)**

An extension to the servlet functionality that enables a simple programmatic interface to Web pages. JSPs are HTML pages with special tags and embedded Java code that is executed on the Web or application server providing dynamic

functionality to HTML pages. JSPs are actually compiled into servlets when first requested and run in the server's JVM.

### **Java virtual machine (JVM)**

The Java interpreter that converts the compiled Java bytecode into the machine language of the platform and runs it. JVMs can run on a client, in a browser, in a middle tier, on a Web, on an application server such as OAS, or in a database server such as Oracle 8i.

### **JDBC**

See Java Database Connectivity.

### **JDeveloper**

Oracle's Java IDE that enables application, applet, and servlet development and includes an editor, compiler, debugger, syntax checker, help system, etc. In version 3.1, JDeveloper has been enhanced to support XML-based development by including the Oracle XDK for Java integrated for easy use along with XML support in its editor.

### **JDK**

See Java Developer's Kit.

### **JServer**

The Java Virtual Machine that runs within the memory space of the Oracle8i database. In Oracle 8i Release 1 the JVM was Java 1.1 compatible while Release 2 is Java 1.2 compatible.

### **JVM**

See Java virtual machine.

### **LAN**

See local area network.

### **local area network (LAN)**

A computer communication network that serves users within a restricted geographical area. LANs consist of servers, workstations, communications hardware (routers, bridges, network cards, etc.) and a network operating system.

### **listener**

A separate application process that monitors the input process.

**Large Object (LOB)**

The class of SQL data type that is further divided into Internal LOBs and External LOBs. Internal LOBs include BLOBs, CLOBs, and NCLOBs while External LOBs include BFILES. See also BFILES, Binary Large Object, Character Large Object.

**LOB**

See Large Object.

**namespace**

The term to describe a set of related element names or attributes within an XML document. The namespace syntax and its usage is defined by a W3C Recommendation. For example, the <xsl:apply-templates/ > element is identified as part of the XSL namespace. Namespaces are declared in the XML document or DTD before they are used by using the following attribute syntax:-  
xmlns:xsl="http://www.w3.org/TR/WD-xsl".

**NCLOB**

See national character Large Object.

**node**

In XML, the term used to denote each addressable entity in the DOM tree.

**national character Large Object**

The LOB datatype whose value is composed of character data corresponding to the database national character set.

**NOTATION**

In XML, the definition of a content type that is not part of those understood by the parser. These types include audio, video, and other multimedia.

**N-tier**

The designation for a computer communication network architecture that consists of one or more tiers made up of clients and servers. Typically two-tier systems are made up of one client level and one server level. A three-tier system utilizes two server tiers, typically a database server as one and a Web or application server along with a client tier.

**OAG**

Open Applications Group.

**OAI**

Oracle Applications Integrator. Runtime with Oracle iStudio development tool that provides a way for CRM applications to integrate with other ERP systems besides Oracle ERP. Specific APIs must be "message enabled." It uses standard extensibility hooks to generate or parse XML streams exchanged with other application systems. In development.

**OAS**

See Oracle Application Server.

**OASIS**

See Organization for the Advancement of Structured Information.

**Object View**

A tailored presentation of the data contained in one or more object tables or other views. The output of an Object View query is treated as a table. Object Views can be used in most places where a table is used.

**object-relational**

The term to describe a relational database system that can also store and manipulate higher-order data types, such as text documents, audio, video files, and user-defined objects.

**Object Request Broker (ORB)**

Software that manages message communication between requesting programs on clients and between objects on servers. ORBs pass the action request and its parameters to the object and return the results back. Common implementations are CORBA and EJBs. See also CORBA.

**OE**

Oracle Exchange.

**Oracle Application Server (OAS)**

The Oracle server that integrates all the core services and features required for building, deploying, and managing high-performance, n-tier, transaction-oriented Web applications within an open standards framework.

**Oracle Integration Server (OIS)**

The Oracle server product that serves as the messaging hub for application integration. OIS contains an Oracle 8i database with AQ and Oracle Workflow and

interfaces to applications using Oracle Message Broker to transport XML-formatted messages between them.

### **ORACLE\_HOME**

The operating system environmental variable that identifies the location of the Oracle database installation for use by applications.

### **OIS**

See Oracle Integration Server.

### **ORB**

See Object Request Broker.

### **Organization for the Advancement of Structured Information (OASIS)**

An organization of members chartered with promoting public information standards through conferences, seminars, exhibits, and other educational events. XML is a standard that OASIS is actively promoting as it is doing with SGML.

### **parent element**

An element that surrounds another element, which is referred to as its child element. For example, <Parent><Child></Child></Parent> illustrates a parent element wrapping its child element.

### **parser**

In XML, a software program that accepts as input an XML document and determines whether it is well-formed and, optionally, valid. The Oracle XML Parser supports both SAX and DOM interfaces.

### **Parsed Character Data (PCDATA)**

The element content consisting of text that should be parsed but is not part of a tag or nonparsed data.

### **PCDATA**

See Parsed Character Data.

### **PDA's**

Personal Digital Assistants, such as Palm Pilot.

### **RDF**

Resource Definition Framework.



**PL/SQL**

The Oracle procedural database language that extends SQL to create programs that can be run within the database.

**prolog**

The opening part of an XML document containing the XML declaration and any DTD or other declarations needed to process the document.

**PUBLIC**

The term used to specify the location on the Internet of the reference that follows.

**renderer**

A software processor that outputs a document in a specified format.

**result set**

The output of a SQL query consisting of one or more rows of data.

**root element**

The element that encloses all the other elements in an XML document and is between the optional prolog and epilog. An XML document is only permitted to have one root element.

**SAX**

See Simple API for XML.

**Simple API for XML (SAX)**

An XML standard interface provided by XML parsers and used by event-based applications.

**schema**

The definition of the structure and data types within a database. It can also be used to refer to an XML document that support the XML Schema W3C recommendation.

**servlet**

A Java application that runs in a server, typically a Web or application server, and performs processing on that server. Servlets are the Java equivalent to CGI scripts.

**session**

The active connection between two tiers.

**SGML**

See Structured Generalized Markup Language.

**Structured Generalized Markup Language (SGML)**

An ISO standard for defining the format of a text document implemented using markup and DTDs.

**Structured Query Language (SQL)**

The standard language used to access and process data in a relational database.

**Server-side Include (SSI)**

The HTML command used to place data or other content into a Web page before sending it to the requesting browser.

**Secure Sockets Layer (SSL)**

The primary security protocol on the Internet, which utilizes a public key/private key form of encryption between browsers and servers.

**SQL**

See Structured Query Language.

**SSI**

See Server-side Include.

**SSL**

See Secure Sockets Layer.

**Stylesheet**

In XML, the term used to describe an XML document that consists of XSL processing instructions used by an XSL processor to transform or format an input XML document into an output one.

**SYSTEM**

The term used to specify the location on the host operating system of the reference that follows.

**tag**

A single piece of XML markup that delimits the start or end of an element. Tags start with < and end with >. In XML, there are start-tags (<name>), end-tags (</name>), and empty tags (<name/>).

**TCP/IP**

See Transmission Control Protocol/Internet Protocol.

**thread**

In programming, a single message or process execution path within an operating system that supports multiple operating systems, such as Windows, UNIX, and Java.

**Transmission Control Protocol/Internet Protocol (TCP/IP)**

The communications network protocol that consists of the TCP which controls the transport functions and IP which provides the routing mechanism. It is the standard for Internet communications.

**Transviewer**

The Oracle term used to describe the Oracle XML Java Beans included in the XDK for Java. These beans include an XML Source View Bean, Tree View Bean, DOMParser Bean, Transformer Bean, and a TransViewer Bean.

**user interface (UI)**

The combination of menus, screens, keyboard commands, mouse clicks, and command language that defines how a user interacts with a software application.

**Uniform Resource Identifier (URI)**

The address syntax that is used to create URLs and XPath.

**Uniform Resource Locator (URL)**

The address that defines the location and route to a file on the Internet. URLs are used by browsers to navigate the World Wide Web and consist of a protocol prefix, port number, domain name, directory and subdirectory names, and the file name. For example <http://technet.oracle.com:80/tech/xml/index.htm> specifies the location and path a browser will travel to find OTN's XML site on the World Wide Web.

**URI**

See Uniform Resource Identifier.

**URL**

See Uniform Resource Locator.

**valid**

The term used to refer to an XML document when its structure and element content is consistent with that declared in its referenced or included DTD.

**W3C**

See World Wide Web Consortium (W3C).

**WAN**

See wide area network.

**Web Request Broker (WRB)**

The cartridge within OAS that processes URLs and sends them to the appropriate cartridge.

**well-formed**

The term used to refer to an XML document that conforms to the syntax of the XML version declared in its XML declaration. This includes having a single root element, properly nested tags, and so forth.

**wide area network (WAN)**

A computer communication network that serves users within a wide geographic area, such as a state or country. WANs consist of servers, workstations, communications hardware (routers, bridges, network cards, etc.), and a network operating system.

**Working Group (WG)**

The committee within the W3C that is made up of industry members that implement the recommendation process in specific Internet technology areas.

**World Wide Web Consortium (W3C)**

An international industry consortium started in 1994 to develop standards for the World Wide Web. It is located at [www.w3c.org](http://www.w3c.org).

**Wrapper**

The term describing a data structure or software that wraps around other data or software, typically to provide a generic or object interface.

**XML Developer's Kit (XDK)**

The set of libraries, components and utilities that provide software developers with the standards-based functionality to XML-enable their applications. In the case of the Oracle XDK for Java, the kit contains an XML Parser, XSL Processor, XML Class Generator, the Transviewer Java Beans and the XSQL Servlet.

**XLink**

The XML Linking language consisting of the rules governing the use of hyperlinks in XML documents. These rules are being developed by the XML Linking Group under the W3C recommendation process. This is one of the three languages XML supports to manage document presentation and hyperlinks (XLink, XPointer, and XPath).

**XML**

See eXtensible Stylesheet Language.

**XML query**

The W3C's effort to create a standard for the language and syntax to query XML documents.

**XML schema**

The W3C's effort to create a standard to express simple data types and complex structures within an XML document. It addresses areas currently lacking in DTDs, including the definition and validation of data types. Oracle XML Schema Processor automatically ensures validity of XML documents and data used in e-business applications, including online exchanges. It adds simple and complex datatypes to XML documents and replaces DTD functionality with an XML Schema definition XML document.

**XPath**

The open standard syntax for addressing elements within a document used by XSL and XPointer. XPath is currently a W3C recommendation. It specifies the data model and grammar for navigating an XML document utilized by XSLT, XLink and XML Query.

**XPointer**

The term and W3C recommendation to describe a reference to an XML document fragment. An XPointer can be used at the end of an XPath-formatted URI. It specifies the identification of individual entities or fragments within an XML document using XPath navigation.

**XSL**

See eXtensible Stylesheet Language.

**XSLFO**

See eXtensible Stylesheet Language Formatting Object.

**XSLT**

See eXtensible Stylesheet Language Transformation.

**XSQL**

The designation used by the Oracle Servlet providing the ability to produce dynamic XML documents from one or more SQL queries and optionally transform the document in the server using an XSL stylesheet.

---

---

# Index

## A

---

- access control
  - making services visible to an application, 18-6
- Adding New Recipients After Enqueue, 9-42
- adding XML document as a child, 20-81
- Advanced Queuing, definition, 9-2
- aggregating XML data
  - other methods, 5-76
  - rollup, see aggregating XML data, 5-76
  - WINDOWING function, 5-76
- ampersand from character data, obtaining, 20-89
- API, Glossary-1
- application profile registry, 18-6
- Application Program Interface,
  - definition, Glossary-1
- application server, Glossary-1
- applications, 2-17
  - communicating XML documents, 2-12
- applying in XML
  - multiple report definitions, 14-7
  - report definition in PL/SQL, 14-8
  - report definition stored in a file, 14-8
  - report definition stored in memory, 14-8
- AQ scenario, 9-2
- AQ XML documents, 9-11
- AQXMLServlet, iDAP, 9-33
- asynchronous parsing, 23-5
- attribute, definition, Glossary-1
- Auditing, 9-5
- authored XML, 2-2
- AUTO\_SECTION\_GROUP
  - how to use, 8-10
- automatic population, 22-30

## B

---

- B2B
  - data exchange, 14-2
  - definition, Glossary-2
  - messaging, 2-25, 2-27, 2-30
- B2C
  - definition, Glossary-2
  - messaging, 2-25
- batch
  - modifications to reports, 14-12
- BC4J
  - advantages, 12-3
  - building XML applications, 12-4
  - building XSQL clients, 12-5
  - definition, Glossary-1
  - features, 12-2
  - flexible deployment, 12-3
  - framework, 12-2
  - JDeveloper, 12-2
  - XSQL clients, 11-8, 12-5
- BC4J (Business Components for Java), 11-6, 12-4
- binary data, 20-85
- Binary Large Object, definition, Glossary-2
- binding
  - clearBindValues(), 7-47
  - setBindValue, 7-43
  - values to queries in XSU PL/SQL API, 7-43
- BLOB, definition, Glossary-2
- Built-in Action Handler, 10-68
- Built-in Action Handler, XSQL, 10-68
- Business Components for Java
  - definition, Glossary-1
  - XSQL clients, 11-8, 12-5

Business components for Java (BC4J), 11-6, 12-4  
Business-to-Business, Glossary-2  
Business-to-Consumer, definition, Glossary-2

## C

---

C Parser, 24-1  
    specifications, E-2  
C++ Parser, 26-1  
callback, definition, Glossary-2  
cartridge, definition, Glossary-2  
Cascading Style Sheets, definition, Glossary-3  
cascading style sheets, see CSS, 4-5  
case-sensitivity, parser, 20-60  
CDATA Section, 20-61  
CDATA, definition, Glossary-3  
CGI, definition, Glossary-3  
Channel Definition Format, definition, Glossary-2  
character sets  
    XML Parser for Java, supported by, C-16  
    XML Schema Processor for Java, supported  
        by, 21-6  
Cheat Sheet  
    XDK for C++, F-1  
    XDK for Java, C-1  
    XDK for PL/SQL, G-1  
CHUNK clause  
    of CREATE TABLE, 5-11  
Class Generator  
    definition, Glossary-3  
    for Java, 22-2  
        complexType, 22-4  
        generate() method, 22-5  
        oracg, 22-3  
        SchemaClassGenerator class, 22-5  
        simpleType, 22-4  
        using with DTDs, 22-8  
        XML Schema, 22-4  
    Java FAQs, 22-30  
    XML C++, 28-1  
Class Generators  
    compared, B-4  
    for Java, explained, 22-30  
classes  
    CGXSDElement, 22-7  
        DOMBuilder(), 23-5  
        DTDClassGenerator(), 22-8  
        SchemaClassGenerator(), 22-5  
        setSchemaValidationMode(), 21-8  
        XMLTreeView(), 23-15  
CLASSPATH, 10-16  
    configuring to run XSU, 7-17  
    definition, Glossary-3  
    settings for class generator for Java, 22-31  
clearBindValues(), 7-47  
clearUpdateColumnNames(), 7-50  
client-server, definition, Glossary-3  
CLOB, definition, Glossary-4  
CLOBs, XML in, 29-27  
command line arguments  
    CUSTOMIZE, 14-6, 14-7, 14-12, 14-16  
    REPORT, 14-16  
command line interface  
    oracg, 22-3  
    oraxml, 20-45  
command line utilities  
    oracg, 22-3  
commands  
    RWCLI60, 14-7, 14-16  
    RWRUN60, 14-7, 14-16  
Common Object Request Broker API,  
    definition, Glossary-4  
Common Oracle Runtime Environment,  
    definition, Glossary-4  
communication  
    between service consumer application and  
        Dynamic Services engine, 18-6  
    supported protocols, 18-7  
compound services  
    Dynamic Services, 18-14  
compression of XML, 20-10  
connecting  
    to a database with a thin driver, 7-23  
    to the database, 7-22  
Connection Definitions, 10-17  
CONTAINS operator, 5-30  
content and document management, 2-17  
content management, 2-17  
context, creating one in XSU PL/SQL API, 7-56  
conventional path load, 2-14



- CORBA, definition, Glossary-4
- CORE, definition, Glossary-4
- creating a node, 20-65
- creating context handles
  - getCtx, 7-43
- creating report definitions
  - XML, 14-15
- CSS and XSL, 4-5
- cube, see aggregating XML data, 5-76
- customization file
  - using in OracleAS Reports Services, 14-7
- CUSTOMIZE
  - OracleAS Reports Services, 14-6
- customizing
  - data presentation, 2-17
  - XML report definition, 14-15

## D

---

- DAD, definition, Glossary-4
- data compression, XML Parser for Java, 20-10
- data exchange applications, 2-11
- database
  - XML support in, 1-7
- Database Access Descriptor, definition, Glossary-4
- datagram, definition, Glossary-4
- DB Access Bean, 23-4
- DBMS\_XMLGEN
  - example to generate complex XML, 5-52
  - summary, 5-2
- DBMS\_XMLQuery
  - bind, 7-43
  - cheat sheet, H-24
  - clearBindValues(), 7-47
  - getXMLClob, 7-47
- DBMS\_XMLQuery(), 7-43
- DBMS\_XMLSave, 7-48
  - cheat sheet, H-24
  - deleteXML, 7-49
  - getCtx, 7-48
  - insertXML, 7-49
  - updateXML, 7-49
- DBMS\_XMLSave(), 7-48
- DBUri, 6-5
  - and object references, 6-12

- syntax guidelines, 6-8
- URL specification, 6-7
- XPath expressions in, 6-8
- DBUri-refs, 6-2, 6-4
  - HTTP access, 6-25
  - scenarios, 6-9
  - where it can be used, 6-12
- DBUriType, 6-3
  - examples, 6-16
- DBViewer Bean, 23-4
- Default SQL to XML Mapping, 7-8
- delete
  - using XSU, 7-16, 7-41
- delete processing, 7-41, 7-53
- demos, 1-17
- design issues, 2-11
- development tools, 3-3
- direct-path load, 2-14
- DISABLE STORAGE IN ROW clause
  - of CREATE TABLE, 5-11
- DocType Node, Creating, 20-67
- DOCTYPE, definition, Glossary-4
- document clones in multiple threads, 20-76
- document management, 2-17
- document mapping, 2-8
- Document Object Model, definition, Glossary-5
- Document Type Definition, definition, Glossary-5
- documents
  - C, 3-20
  - C++, 3-22
  - Java, 3-17
  - PL/SQL, 3-24
- DOM
  - API, 20-65
  - definition, Glossary-5
  - interface, 29-6
  - tree-based API, 20-8
  - using API, 29-30
- DOM and SAX APIs, 20-8, 24-9, 26-9
  - guidelines for usage, 20-9
- DOMBuilder Bean, 23-2, 23-5
  - asynchronous parsing, 23-5
- DOMException when Setting Node Value, 20-74
- DOMNamespace() class, 20-23
- domsample, 29-10

- downgrading
  - to Oracle release 8.1, 20-12
- DTD, 20-55
  - caching, 20-57
  - definition, Glossary-5
  - limitations, 21-3
  - using with Class Generator for Java, 22-8

- Dynamic Services
  - client library, 18-6
  - communication, 18-6
  - compound services, 18-14
  - conditional services, 18-15
  - consumer application example, 18-18
  - developing services, 18-17
  - engine, 18-6
  - failover, 18-14
  - framework, 18-9
  - ICE, 18-11
  - iIntegrating with other Oracle products, 18-16
  - multiple channel capabilities, 18-11
  - OSS, 18-18
  - service registry, 18-6
  - SOAP, 18-11
  - Wireless Edition, 18-11
- Dynamic Services Content Provider Adapter (DSCPA), 19-7

## E

---

- EJB, definition, Glossary-5
- Electronic Data Interchange, definition, Glossary-5
- element, definition, Glossary-5
- elements
  - complexType, 22-4
  - simpleType, 22-4
- empty element, definition, Glossary-5
- ENABLE STORAGE IN ROW clause
  - of CREATE TABLE, 5-11
- Enterprise Java Bean, definition, Glossary-5
- entity, definition, Glossary-6
- errors when parsing a document, 29-38
- errors, HTML, 20-100
- eXtensible Markup Language
  - XML, A-2
- eXtensible Stylesheet Language Formatting Object,

- definition, Glossary-6
- eXtensible Stylesheet Language Transformation,
  - definition, Glossary-6
- eXtensible Stylesheet Language,
  - definition, Glossary-6
- extracting XML, 1-6

## F

---

- factory method, 6-17
- failover services
  - Dynamic Services, 18-14
- FAQ, 3-26
  - JDeveloper, 11-20
  - Oracle Text, 8-50
  - XML applications, 11-26
  - XSU, 7-58
- FAQs, XML and AQ, 9-41
- features, XML, A-6
- first child node's value, 20-70
- Frequently Asked Questions
  - Class Generator for Java, 22-30
  - XML Parser for PL/SQL, 29-20
  - XSQL Servlet, 10-74
- Frequently Asked Questions, XML and AQ, 9-41
- further references, 3-40

## G

---

- generated XML, 2-2, 2-7, 3-26
  - customizing, 7-12
- generating
  - simpleType element classes, 22-7
  - top level complexType element classes, 22-7
- generating XML, 7-17, 7-30
  - using DBMS\_XMLQuery, 7-43
  - using XSU command line, getXML, 7-17
- getCtx, 7-43, 7-48
- getDocType(), 22-8
- getNodeValue(), 29-41
- getXML, 7-17
- getXMLClob, 7-47

## H

---

HASPATH operators, 8-10  
hierarchical mapping, 20-99  
HP/UX, 20-110  
HTML  
    definition, Glossary-7  
    errors, 20-100  
    parsing, 29-39  
HTTP  
    access for DBUri-refs, 6-25  
    definition, Glossary-7  
    to access AQ XML Servlet, 9-33  
http  
    //otn.oracle.com/tech/xml/, 24-2  
HttpUriType, 6-3  
Hub-and-Spoke Architecture, 9-4  
hybrid storage, 2-5  
Hypertext Markup Language,  
    definition, Glossary-7  
Hypertext Transport Protocol,  
    definition, Glossary-7  
hypertext, definition, Glossary-7

## I

---

ICE  
    Dynamic Services and Oracle Syndication  
        Server, 18-11  
iDAP, 9-6  
    AQ XML Schemas, 9-33  
    AQXMLServlet, deploying and creating, 9-33  
    architecture, 9-6  
    HTTP used to access AQXMLServlet, 9-33  
    interface explained, 9-6  
    payload or method invocation, 9-10  
    SMTP, 9-8  
IDE, definition, Glossary-7  
IIOP, definition, Glossary-7  
Information and Content Exchange (ICE)  
    protocol, 19-4  
INPATH operator, 8-10  
insert, XSU, 7-14  
inserting XML  
    using XSU, 7-36

insertXML, 7-49  
install  
    Oracle Text, 8-4  
installing  
    class generator for Java, 22-30  
instantiate, definition, Glossary-8  
Integrated Development Environment,  
    definition, Glossary-7  
Integrated tools, 1-14  
interMedia, 3-16, 8-3  
    CONTAINS operator, 8-6  
    querying, 8-6  
interMedia, definition, Glossary-8  
Internet Data Access Presentation (iDAP), 9-6  
Internet File System, definition, Glossary-7  
Internet-Data-Access-Presentation, see iDAP, 9-6

## J

---

Java Bean, definition, Glossary-8  
Java Beans, 3-9  
Java Class Generator, 22-1  
Java Database Connectivity, definition, Glossary-8  
Java Runtime Environment, definition, Glossary-8  
Java, definition, Glossary-8  
JAVASYSPRIV, granting, 20-96  
JDBC driver, 7-22  
JDBC, definition, Glossary-8, Glossary-9  
JDeveloper, 11-1, 12-1  
    3.2, 11-2  
    BC4J, 12-2  
    definition, Glossary-9  
    FAQ, 11-26  
    introduction, 11-2  
    mobile application, 11-20  
    support for XDK for Java Beans, 23-2  
    using XSQL servlet from, 11-17  
    what's needed, 11-5  
    XML data generator web bean, 11-15  
    XML features, 11-11  
JDK, 20-87  
    definition, Glossary-8  
JRE, definition, Glossary-8  
JServer(JVM) Option, 29-25  
JServer, definition, Glossary-9

JSP, definition, Glossary-8

JVM, 29-25

definition, Glossary-9

## K

---

keepObjectOpen(), 7-28, 7-45

## L

---

LAN, definition, Glossary-9

Linux, 29-31

listener, definition, Glossary-9

loading XML documents, 2-14

LOB, definition, Glossary-10

LOBFILE, syntax, 2-14

LOBs

in-line storage, 5-11

number of bytes manipulated in, 5-11

local area network, definition, Glossary-9

## M

---

management

content and document, 2-17

mapping

hierarchical, 20-99

primer, XSU, 7-8

maxRows, 7-27

memory errors, 29-27

Merging XML Documents, 20-94

Message Retention, 9-5

message server, 9-3

message transformation, XML AQ, 9-38

messaging

B2B and B2C, 2-25

method

getDocument(), DOMBuilder Bean, 23-6

methods

addXSLTransformerListener(), 23-11

DOMBuilder Bean, 23-5

domBuilderError(), 23-6

DOMBuilderOver(), 23-6

domBuilderStarted(), 23-6

generate(), 22-5, 22-8

getDocType(), 22-8

getPreferredSize(), TreeViewer Bean  
(XML), 23-15

setType, 22-6

setXMLDocument(doc), 23-15

updateUI(), TreeViewer Bean (XML), 23-15

Mining, 9-5

mobile application

JDeveloper, 11-20

mode, definition, Glossary-10

multiple outputs, 20-109

multiple XML documents, delimiting, 20-92

MULTISET operator

using with SYS\_XMLGEN selects, 5-67

## N

---

namespace

feature in XML Class Generator for Java, 22-4

namespaces

XML, 20-5, A-4

xmlns, 4-3

national character Large Object,

definition, Glossary-10

no rows exception, 7-33

Non-SAX Callback Functions, E-13

NOTATION, definition, Glossary-10

N-tier, definition, Glossary-10

## O

---

OAG, definition, Glossary-10

OAI, definition, Glossary-11

OAS, definition, Glossary-11

OASIS, definition, Glossary-12

object references and DBUri, 6-12

Object View, definition, Glossary-11

object-relational, definition, Glossary-11

OE, definition, Glossary-11

OIS, definition, Glossary-11

OMB, 16-4

Open Applications Group, definition, Glossary-10

operators

HASPATH, 8-10

INPATH, 8-10

- ora
  - node-set, 20-50
  - output, 20-50
- oracg, 22-3
- oracg command line utility, 22-3
- Oracle Application Server, definition, Glossary-11
- Oracle Exchange
  - ATP, 16-5
  - definition, Glossary-11
  - OMB, 16-4
  - transactions use xml formats, 16-2
  - webMethods, 16-4
  - XML delivery formats, 16-4
  - XML Message Designer, 16-9
- Oracle Integration Server, definition, Glossary-11
- Oracle interMedia, 3-16
- Oracle Internet Directory server, 18-6
- Oracle Syndication Server (OSS)
  - architecture, 19-5
  - content providers, 19-7
  - content subscribers, 19-7
  - Dynamic Services, 18-18
  - ICE protocol, 19-4
- Oracle Syndication Services
  - Dynamic Services Content Provider Adapter (DSCPA), 19-7
- Oracle Text, 8-3
  - CONTAINS and XMLType, 5-30
  - users and roles, 8-5
- Oracle XML, 1-2
- Oracle XML Parsers, comparison, B-2
- ORACLE\_HOME, definition, Glossary-12
- Oracle9i Reports Services
  - report definitions at runtime, 14-6
- OracleAS Reports Services, 14-1
  - batch report modifications, 14-12
  - customizing reports at runtime, 14-8
- OracleText
  - query applications, 8-23
  - querying, 8-29
- OracleXML
  - putXML, 7-20
  - XSU command line, 7-17
- OracleXMLNoRowsException, 7-57
- OracleXMLQuery, 7-20

- OracleXMLSave, 7-20, 7-35, 7-36, 7-38, 7-41
- OracleXMLSQLException, 7-56
- OraDBUriServlet
  - installing, 6-27
  - security, 6-26
- OraDbUriServlet
  - servlet mechanism, 6-25
- oraxml, 20-45
- oraxsl, 20-46
  - command line interfaces
    - oraxsl, 20-46
- OraXSL Parser, 20-99
- ORB, definition, Glossary-11
- out of memory errors, 29-27
- Out Variable, 10-79
- Output Escaping, 20-92

## P

---

- paginating results, 7-27
- parent element, definition, Glossary-12
- parser case-sensitivity, 20-60
- Parser for C, 24-1
  - specifications, E-2
- Parser for C++, 26-1
- Parser for Java, 20-1
  - constructor extension functions, 20-48
  - oraxsl command line interfaces
    - oraxsl, 20-46
  - return value extension function, 20-49
  - validation modes, 20-5
- Parser for PL/SQL, 29-1
- parser, definition, Glossary-12
- parsers
  - uninstalling, 20-83
- Parsers, XML, 20-2
- parsing
  - errors, 29-38
  - HTML, 29-39
  - string, 20-88
  - URLs, 29-38
- PATH\_SECTION\_GROUP
  - how to use, 8-10
- PCDATA, definition, Glossary-12
- PCTVERSION parameter

- of CREATE TABLE, 5-11
- performing batch modifications in XML, 14-12
- Personal Digital Assistant, definition, Glossary-12
- PL/SQL
  - binding values in XSU, 7-47
  - definition, Glossary-13
  - generating XML with DBMS\_XMLQuery, 7-43
  - parser, 29-1
  - XSU, 7-43
- point-to-point, 9-2
- processing
  - delete, 7-53
  - insert, 7-36
  - insert in PL/SQL, 7-49
  - update, 7-38, 7-51
- prolog, definition, Glossary-13
- properties
  - setGeneratorComments(), 22-8
  - setJavaPackage(string), 22-8
  - setOutputDirectory(string), 22-8
- PUBLIC, definition, Glossary-13
- publish/subscribe, 9-2
- putXML, 7-19

## Q

---

- query
  - results, 8-41
- query application, 8-29
- query, XML, A-4
- querying
  - XML documents indexed with OracleText, 8-24
- queues
  - XMLType, 9-37

## R

---

- recommendations, W3C, A-4
- renderer, definition, Glossary-13
- reports, generating in XML, 14-1
- Resource Definition Framework,
  - definition, Glossary-12
- result set objects, 7-30
- result set, definition, Glossary-13
- roadmap, 3-xlv

- root element, definition, Glossary-13
- root objects, creating multiple with class
  - generator, 22-31
- running
  - XML
    - report definition by itself, 14-16
    - report definitions, 14-16
- runtime
  - customization
    - XML report definition, 14-15

## S

---

- samples, 1-17
- SAX, 20-2
  - event-based API, 20-8
- SAX API, 20-8, 20-68, 24-9, 26-9, F-17
- SAX API Function, E-13
- SAX Functions, E-13
- SAX, definition, Glossary-13
- SAXNamespace() Class, 20-41
- SAXParser() class, 20-27
- SAXSample.java, 20-69
- schema, definition, Glossary-13
- Schema, XML, definition, 20-86
- SchemaClassGenerator, 22-5
- search
  - XML documents, 8-17
- section preference
  - creating a, Oracle Text index, 8-18
- section\_group
  - deciding which to use, Oracle Text, 8-19
- security
  - OraDBUriServlet, 6-26
- select
  - with XSU, 7-14
- sending XML data, 2-11
- Server-side Include, definition, Glossary-14
- services
  - compound, Dynamic Services, 18-14
  - conditional, Dynamic Services, 18-15
  - failover, Dynamic Services, 18-14
- Servlet Conditional Statements, 10-74
- servlet, definition, Glossary-13
- servlet, XSQL, 10-1

- session, definition, Glossary-13
- setBindValue, 7-43
- setKeyColumn, 7-42
- setKeyColumn(), 7-54
- setMaxRows, 7-45
- setRaiseNoRowsException(), 7-45
- setSkipRows, 7-45
- setStylesheetHeader(), 7-46
- setUpdateColumnName(), 7-50, 7-53
- setUpdateColumnNames()
  - XML SQL Utility (XSU)
    - setUpdateColumnNames(), 7-40
- SGML, definition, Glossary-14
- Simple API for XML, definition, Glossary-13
- simpleType, 22-4
  - generating element class, 22-7
- skipRows, 7-27
- SMTP
  - iDAP, 9-8
- SOAP, 18-11
- special characters, 20-89
- SQL\*Loader
  - conventional path load, 2-14
  - direct-path load, 2-14
  - LOBFILE, 2-14
- SQL, definition, Glossary-14
- SSI, definition, Glossary-14
- SSL, definition, Glossary-14
- storing XML, 1-6, 7-35
  - using XSU command line, putXML, 7-19
- storing XML in the database, 7-48
- Stylesheet, definition, Glossary-14
- stylesheets
  - template processing, F-16
  - XSLT, F-16
  - XSU, 7-46
- SYS\_DBURIGEN function, 6-20
  - examples, 6-22
  - inserting database references, 6-22
  - returning partial results, 6-22
  - RETURNING Uri-refs, 6-24
- SYS\_XMLAGG, 5-71
  - aggregating all POs into one XML document, 5-74
  - aggregating XMLType fragments,
    - example, 5-73
  - aggregating XMLType instances stored in tables, 5-73
  - summary, 5-3
- SYS\_XMLGEN
  - summary, 5-3
- SYS\_XMLGEN function
  - converting a UDT to XML, 5-66
  - converting XMLType instances, 5-67
  - generating XML in SQL queries, 5-10
  - static member function create, 5-65
  - using with object views, 5-69
  - XMLGenFormatType object, 5-63
- SYSTEM, definition, Glossary-14
- System.out.println(), 20-89
- SYS.UriFactoryType, 6-3

---

## T

- tag, definition, Glossary-15
- TCP/IP, definition, Glossary-15
- thin driver
  - connecting XSU, 7-23
- thread safety, 26-3
- thread, definition, Glossary-15
- Tracking, 9-5
- transactions
  - inbound in Oracle Exchange, 16-2
  - outbound in Oracle Exchange, 16-2
  - pass through in Oracle Exchange, 16-2
  - stored, in Oracle Exchange, 16-2
- transformations, 2-7
- Transviewer Beans, 23-1
- Transviewer, definition, Glossary-15
- Treeviewer Bean, 23-3, 23-13
- Tuning with XSQL, 10-54

---

## U

- UI, definition, Glossary-15
- Uniform Resource Identifier,
  - definition, Glossary-15
- Uniform Resource Locator, definition, Glossary-15
- uninstalling parsers, 20-83
- update processing, 7-51

- update, XSU, 7-15
- updating
  - table using keyColumns, XSU, 7-39
  - using XSU, 7-38
- upgrading
  - scripts
    - XMLU815.SQL, 20-11
    - XMLU816.SQL, 20-11
    - XMLU817.SQL, 20-11
  - XDK for Java, 20-11
  - XDK for Java to Oracle9i, 20-11
  - XML, 20-11
- URI, definition, Glossary-15
- UriFactory package, 6-17
  - configuring to handle DBUri-ref, 6-36
  - factory method, 6-17
  - registering ecom protocol, 6-18
- Uri-ref, see also Uri-reference, 6-2
- Uri-reference
  - database and session, 6-12
  - datatypes, 6-3
  - DBUri, 6-5
  - DBUri and object references, 6-12
  - DBUri syntax guidelines, 6-8
  - DBUri-ref, 6-2, 6-4
  - DBUri-ref uses, 6-12
  - DBUriType, 6-3
  - DBUriType examples, 6-16
  - explained, 6-2
  - HTTP access for DBUri-ref, 6-25
  - HttpUriType, 6-3
  - UriFactory package, 6-17
  - UriType, 6-3
  - URIType examples, 6-15
  - URITypes, 6-14
- URIType, 6-14
  - examples, 6-15
- UriTypes, 6-3
  - benefits, 6-4
  - summarized, 5-3
- URL, definition, Glossary-15
- usage techniques, 7-56
- user interface, definition, Glossary-15
- useStyleSheet(), 7-47
- UTF-16 Encoding, 20-79

## V

---

- valid, definition, Glossary-16
- validating against XML schema, 20-85
- validation
  - non-validating mode, 20-5
  - partial validation mode, 20-5
  - schema validation mode, 20-5
  - validating Mode, 20-5
- value of a tag, obtaining, 20-96

## W

---

- W3C DOM API, G-11
- W3C XML recommendations, A-4
- W3C, definition, Glossary-16
- WAN, definition, Glossary-16
- Web bean
  - XML data generator, 11-15
- Web Request Broker, definition, Glossary-16
- Web to database, 2-11
- webMethods, 16-4, 16-5
- well-formed, definition, Glossary-16
- WG, definition, Glossary-16
- why use Oracle8i XML?, 1-11
- wide area network, definition, Glossary-16
- WINDOWING function, see aggregating XML data, 5-76
- Wireless Edition
  - and Dynamic Services, 18-11
- World Wide Web Consortium,
  - definition, Glossary-16
- Wrapper, definition, Glossary-16
- WRB, definition, Glossary-16
- wrong\_document\_err, 20-72

## X

---

- XDK for C, E-1
- XDK for C++, Specifications, F-1
- XDK for Java
  - upgrading, 20-11
- XDK for PL/SQL Toolkit, 29-22
- XDK Version Numbers, 20-86
- XDK, definition, Glossary-17
- XLink, definition, Glossary-17



- XML
  - applying
    - multiple report definitions, 14-7
    - report definition in PL/SQL, 14-8
    - report definition stored in a file, 14-8
    - report definition stored in memory, 14-8
  - authored, 2-2
  - business components for Java, 11-6, 12-4
  - creating
    - report definition, 14-15
  - design issues, 2-11
  - generated, 2-2
  - good references, 20-109
  - Oracle XML, 1-2
  - report definitions
    - applying, 14-6
    - applying via PL/SQL, 14-8
    - batch modifications, 14-12
    - running, 14-16
  - running
    - report definition by itself, 14-16
    - report definitions, 14-16
  - serialization/compression, 20-10
  - upgrading, 20-11
- XML applications, 11-1, 12-1
  - JDeveloper, 11-26
  - with JDeveloper, 11-14
- XML AQ message transformation
  - AQ
    - XML message transformation, 9-38
- XML C++ Class Generator, 28-1
- XML Class Generator, 3-8
  - oracg utility, 22-3
- XML Class Generator for Java, 22-2
- XML Class Generators, compared, B-4
- XML components, 3-2
  - generating XML documents, 3-17
- XML data
  - sending, 2-11
- XML data generator, 11-15
- XML Developer's Kit, definition, Glossary-17
- XML document, added as a child, 20-81
- XML documents, 3-17
  - communicating, 2-12
  - interMedia, 8-17
  - sections, 8-39
- XML Documents, Merging, 20-94
- XML Family, A-5
- XML features, A-6
  - in JDeveloper 3.2, 11-11
- XML in CLOBs, 29-27
- XML Message Designer
  - Oracle Exchange, 16-9
- XML messaging services
  - Oracle Exchange, 16-9
- XML Namespaces, 20-5
- XML namespaces, A-4
- XML Parser
  - oraxml command line interface, 20-45
- XML Parser for C, 24-1
  - sample programs, 24-13
  - specifications, E-2
- XML Parser for C++, 26-1
- XML Parser for Java, 20-1
  - compression
    - XML data, using XML Parser for Java, 20-10
- XML Parser for PL/SQL, 29-1
  - FAQs, 29-20
- XML parsers, 3-6
- XML Parsers and Class Generators,
  - comparing, B-1
- XML query, A-4
- XML query, definition, Glossary-17
- XML Schema, A-4
  - compared to DTD, 21-2
  - DTD limitations, 21-3
  - explained, 21-2
  - features, 21-3
  - iDAP and AQ, 9-33
  - processor for Java
    - how to run the sample program, 21-9
    - supported character sets, 21-6
    - usage, 21-8
  - processor for Java features, Oracle's, 21-6
- XML schema, 2-8
- XML schema, definition, 20-86, Glossary-17
- XML SQL Utility (XSU), 3-14, 7-43
  - advanced techniques, exception handling (PL/SQL), 7-58
  - binding values

- PL/SQL API, 7-47
- clearBindValues() with PL/SQL API, 7-47
- command line usage, 7-17
- connecting to the database, 7-22
- connecting with a thin driver, 7-23
- connecting with OCI\* JDBC driver, 7-22
- customizing generated XML, 7-12
- DBMS\_XMLQuery, 7-43
- DBMS\_XMLSave(), 7-48
- deletes, 7-16
- deleting from XML documents, 7-41
- dependencies and installation, 7-4
- explained, 7-2
- for Java, 7-20
- getXML command line, 7-17
- getXMLClob, 7-47
- how it works, 7-14
- inserting with command line and putXML, 7-19
- inserting XML into database, 7-36
- inserts, 7-14
- keepObjectOpen function, 7-28
- mapping primer, 7-8
- OracleXLIQuery API, 7-20
- OracleXMLSave API, 7-20
- putting XML back in database with
  - OracleXMLSave, 7-35
- selects, 7-14
- setKeycolumn function, 7-42
- setRaiseNoRowsException(), 7-45
- setting stylesheets, PL/SQL, 7-46
- updates, 7-15
- updating, 7-39
- updating XML documents in tables, 7-38
- XML SQL Utility (XSU)
  - useStyleSheet(), 7-47
- XML SQL Utility (XSU)
  - creating context handles with getCtx, 7-43
- XML streams
  - how to become a supplier of live (reports), 14-20
- XML to Java Object Mapping, 22-30
- XML Transviewer Java Beans, 3-9, 23-2
- XML Tree, Traversing, 20-66
- XML, definition, Glossary-6
- XML, loading, 2-14

- XML\_SECTION\_GROUP
  - how to use, 8-8
- XMLAGG, 5-71
- XML-based standards, A-5
- xmlcg usage, 28-4
- XMLGEN, is deprecated. See DBMS\_XMLQUERY and DBMS\_XMLSAVE, 7-4
- XMLGenFormatType object, 5-63
- XMLNode.selectNodes() Method, 20-67
- XMLParser() API, F-9
- XMLSourceView Bean, 23-3, 23-15
- XMLTransformPanel() Bean, 23-3, 23-20
- XMLType
  - CONTAINS operator, 5-30
  - database support, 1-7
  - interaction with other SQL constructs, 5-7
  - queues, 9-37
  - storage characteristics, 5-10
  - summary, 5-2
  - Xpath support, 5-30
- XMLU815.SQL script, 20-11
- XMLU816.SQL script, 20-11
- XMLU817.SQL script, 20-11
- XPath, A-4
  - basics, 4-5
  - definition, Glossary-17
  - support, 5-30
- XPointer, A-4
- XPointer, definition, Glossary-17
- XSL
  - and CSS, 4-5
  - basics, 4-2
  - converting a string to a nodeset, 4-13
  - converting a tag to a link in HTML, 4-16
  - converting XML to HTML, 4-7
  - ensuring the DTD file can be located, 4-18
  - error XSL-1009 attribute 'XSL Version' not found in HTML, 4-21
  - frequently asked questions, 4-6
  - good references, 20-109
  - IF statement, 4-6
  - passing a parameter from a Java program to a stylesheet, 4-20
  - preventing namespace definition from being repeated, 4-19

- selecting specific attributes, 4-7
  - specifying NULL indicators, 4-10
  - transferring tag names, 4-10
  - using XSL headers in WML
    - transformations, 4-17
  - working with whitespace, 4-8
- XSL stylesheets
  - in reports, 14-5
  - setStyleSheetHeader() in XSU PL/SQL, 7-46
  - useStyleSheet() with XSU PL/SQL, 7-47
- XSL Transformation (XSLT) Processor, 3-7, 20-4
- XSL, definition, Glossary-6
- XSLFO, definition, Glossary-6
- xslsample, 29-11
- XSLT, 20-4
  - 1.1 specification, 4-4
  - explained, 4-4
  - ora
    - node-set built in extension, 20-50
    - output built in extension, 20-50
  - XSLTransformer bean, 23-9
- XSLT API, F-15
- XSLT API Functions, E-12
- XSLT Processor, 29-6
- XSLT Processor API, G-9
- XSLT, definition, Glossary-6
- XSLTransformer Bean, 23-3, 23-9
- XSQL
  - action handler errors, 10-72
  - built-in action handler elements, 10-68
  - clients, building with BC4J, 12-5
  - pluggable data source in reports, 14-19
- XSQL Clients with BC4J, 12-5
- XSQL Page Processor, 3-10
- XSQL servlet, 3-10, 10-1, 11-17
  - FAQs, 10-74
- XSQL, definition, Glossary-18
- XSQLCommandLine Utility, 10-18
- XSQLConfig.xml, 10-54
- XSU, 3-14
  - client-side, 7-17
  - FAQ, 7-58
  - generating XML, 7-17
  - generating XML strings from a table,
    - example, 7-22
    - insert processing in PL/SQL, 7-49
    - mapping primer, 7-8
    - PL/SQL, 7-43
    - stylesheets, 7-46
    - usage guidelines, 7-8
    - using, 7-2
    - where you can run, 7-5

